How ClassLoader Works in Java? Example

Java class loaders are used to load classes at runtime. ClassLoader in Java works on three principles: `delegation`, `visibility`, and `uniqueness`. Delegation principle forward request of class loading to parent class loader and only loads the class if the parent is not able to find or load the class. The visibility principle allows the child class loader to see all the classes loaded by the parent ClassLoader, but the parent class loader can not see classes loaded by a child. The uniqueness principle allows one to load a class exactly once, which is basically achieved by delegation and ensures that child ClassLoader doesn't reload the class already loaded by a parent. Correct understanding of class loader is a must to resolve issues like NoClassDefFoundError in Java and java.lang.ClassNotFoundException, which is related to class loading.

ClassLoader is also an important topic in advanced Java Interviews, where a good knowledge of the working of Java ClassLoader and How classpath works in Java is expected from Java programmers.

I have always seen questions like, **Can one class be loaded by two different ClassLoader in Java** on various Java Interviews. In this Java programming tutorial, we will learn what is ClassLoader in Java, How ClassLoader works in Java and some specifics about Java ClassLoader.

What is ClassLoader in Java

ClassLoader in Java is a class that is used to load class files in Java. Java code is compiled into a class file by javac compiler and JVM executes the Java program, by executing byte codes written in the class file. ClassLoader is responsible for loading class files from file systems, networks, or any other source.

There is three default class loader used in Java, **Bootstrap**, **Extension,** and **System or Application class loader**.

Every class loader has a predefined location, from where they load class files. The bootstrap class loader is responsible for loading standard JDK class files from `rt.jar` and it is the parent of all class loaders in Java.

The bootstrap class loader doesn't have any parents if you call `String.class.getClassLoader()` it will return `null and any code based on that may throw` NullPointerException in Java. The bootstrap class loader is also known as **Primordial ClassLoader** in Java.

Extension ClassLoader delegates class loading request to its parent, `Bootstrap,` and if unsuccessful, loads class form `jre/lib/ext` directory or any other directory pointed by `java.ext.dirs` system property. Extension ClassLoader in JVM is implemented by `sun.misc.Launcher$ExtClassLoader.`

The third default class loader used by JVM to load Java classes is called System or Application class loader and it is responsible for loading application-specific classes from CLASSPATH environment variable, `-classpath` or `-cp` command line option, `Class-Path` attribute of Manifest file inside JAR.

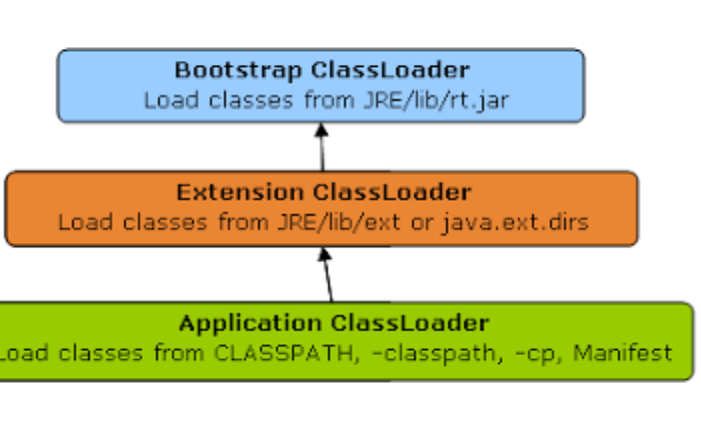
Application class loader is a child of Extension ClassLoader and its implemented by `sun.misc.Launcher$AppClassLoader` class. Also, except for the Bootstrap class loader, which is implemented in the native language mostly in C, all Java class loaders are implemented using `java.lang.ClassLoader`.

In short here is the location from which Bootstrap, Extension, and Application ClassLoader load Class files.

1) Bootstrap ClassLoader - JRE/lib/rt.jar

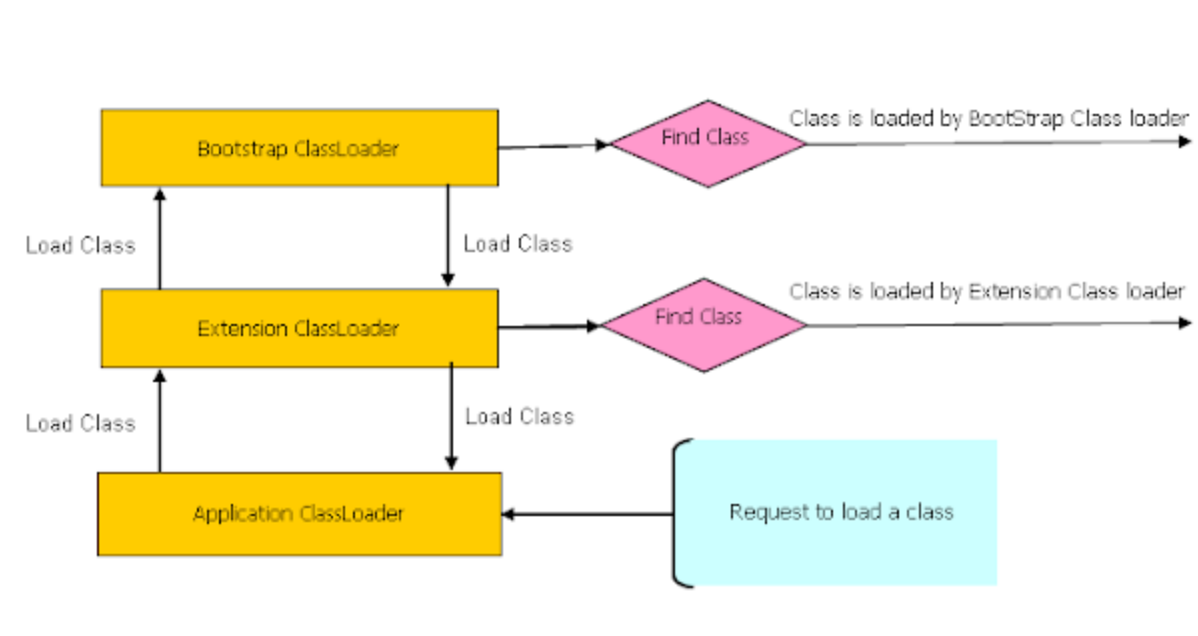
2) Extension ClassLoader - JRE/lib/ext or any directory denoted by `java.ext.dirs`

3) Application ClassLoader - `CLASSPATH` environment variable, `-classpath` or `-cp` option, Class-Path attribute of Manifest inside JAR file.



How ClassLoader works in Java?

As I explained earlier Java ClassLoader works in three principles: `delegation`, `visibility`, and `uniqueness`. In this section, we will see those rules in detail and understand the working of Java ClassLoader with example. By the way here is a diagram that explains How ClassLoader loads class in Java using delegation.



1. Delegation principles

As discussed on when a class is loaded and initialized in Java, a class is loaded in Java, when it's needed. Suppose you have an application-specific class called `Abc.class`, the first request of loading this class will come to Application ClassLoader which will delegate to its parent Extension ClassLoader which further delegates to `Primordial` or `Bootstrap` class loader.

Primordial will look for that class in `rt.jar` and since that class is not there, a request comes to Extension class loader which looks on `jre/lib/ext` directory and tries to locate this class there, if the class is found there then Extension class loader will load that class and Application class loader will never load that class but if it's not loaded by extension class-loader then Application class loader loads it from Classpath in Java. Remember `Classpath` is used to load class files while PATH is used to locate executables like javac or java command.

2. Visibility Principle

According to the visibility principle, Child ClassLoader can see class loaded by Parent ClassLoader but vice-versa is not true. This means if class `Abc` is loaded by Application class loader than trying to load class ABC explicitly using extension ClassLoader will throw either java.lang.ClassNotFoundException. as shown in the below Example

```
package test;

import java.util.logging.Level;
import java.util.logging.Logger;

/**
 * Java program to demonstrate How ClassLoader works in Java,
 * in particular about the visibility principle of ClassLoader.
 *
 * @author Javin Paul
 */

public class ClassLoaderTest {

    public static void main(String args[]) {
        try {
            //printing ClassLoader of this class
System.out.println("ClassLoaderTest.getClass().getClassLoader() : "
                                 + ClassLoaderTest.class.getClassLoader());


            //trying to explicitly load this class again using Extension class loader
            Class.forName("test.ClassLoaderTest", true
                            ,  ClassLoaderTest.class.getClassLoader().getParent());
        } catch (ClassNotFoundException ex) {
Logger.getLogger(ClassLoaderTest.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

}

Output:
ClassLoaderTest.getClass().getClassLoader() : sun.misc.Launcher$AppClassLoader@601bb1
16/08/2012 2:43:48 AM test.ClassLoaderTest main
SEVERE: null
java.lang.ClassNotFoundException: test.ClassLoaderTest
        at java.net.URLClassLoader$1.run(URLClassLoader.java:202)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(URLClassLoader.java:190)
        at
sun.misc.Launcher$ExtClassLoader.findClass(Launcher.java:229)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:306)
        at java.lang.ClassLoader.loadClass(ClassLoader.java:247)
        at java.lang.Class.forName0(Native Method)
        at java.lang.Class.forName(Class.java:247)
        at test.ClassLoaderTest.main(ClassLoaderTest.java:29)
```

3. Uniqueness Principle

According to this principle, a class loaded by a Parent should not be loaded by Child ClassLoader again. Though it's completely possible to write a class loader that violates Delegation and Uniqueness principles and loads class by itself, it's not something which is beneficial. You should follow all class loader principles while writing your own ClassLoader.

How to load class explicitly in Java

Java provides API to explicitly load a class by `Class.forName(classname)` and `Class.forName(classname, initialized, classloader)`, remember JDBC code which is used to load JDBC drives we have seen in Java program to Connect Oracle database.

As shown in the above example you can pass the name of ClassLoader which should be used to load that particular class along with the binary name of the class. Class is loaded by calling `loadClass()` method of `java.lang.ClassLoader` class which calls `findClass()` method to locate bytecodes for the corresponding class.

In this example, Extension ClassLoader uses `java.net.URLClassLoader` which searches for class files and resources in JAR and directories. any search path which is ended using "/" is considered a directory.

If `findClass()` does not found the class then it throws java.lang.ClassNotFoundException and if it finds it calls `defineClass()` to convert bytecodes into a .class instance which is returned to the caller.

Where to use ClassLoader in Java?

ClassLoader in Java is a powerful concept and is used in many places. One of the *popular examples of ClassLoader* is `AppletClassLoader` which is used to load a class by `Applet` since `Applets` are mostly loaded from the internet rather than a local file system.

By using separate ClassLoader you can also load the same class from multiple sources and they will be treated as different classes in JVM. J2EE uses multiple class loaders to load a class from a different location like classes from the WAR file will be loaded by Web-app ClassLoader while classes bundled in EJB-JAR are loaded by another classloader.

Some web server also supports hot deploy functionality which is implemented using ClassLoader. You can also use ClassLoader to load classes from a database or any other persistent store.

That's all about **What is ClassLoader in Java** and **How ClassLoader works in Java**. We have seen delegation, visibility, and uniqueness principles which are quite important to debug or troubleshoot any ClassLoader related issues in Java. In summary knowledge of How ClassLoader works in Java is a must for any Java developer or architect to design Java applications and packaging.

Other **Java Tutorials** from Javarevisited you may like
How HashMap works in Java
How volatile variable works in Java
Why multiple inheritance is not supported in Java
Top 10 JDBC best practices for Java programmer
10 OOPS design principle Java programmer should know

By javin paul
Labels: core java , core java interview question , JVM Internals

20 comments :

Unknown said...
Hi!
Thanks for the article. I have a question about the sentence: "Java provides API to explicitly load a class by Class.forName(classname) and Class.forName(classname, initialized, classloader), remember JDBC code which is used to load JDBC drives we have seen in Java program to Connect Oracle database."
I cannot find the explicit loading of class in that article. Do I misunderstand something?

December 10, 2012 at 12:25 PM

sajjad paracha said...
Good Article , I would like to add one more exception/error caused by ClassLoading problems. NoSuchMethodError , this happens mostly in web applications or J2EE applications where you have the latest jar file of a utility such as apache common utilities in lib folder of your application but the server has an older version of same jar file.When your application needs the class it is is already loaded by the parent class loader (your server's class loader) but when the application wants to call a method which is available in new jar but not in the old jar loaded by parent class loader you will face NOSuchMethod Error exception. i recently went through all these problems trying to deploy a JPA2.0 application on weblogic10.3.3 server (which is JPA1.0 compliant which means you will have old jar files loaded in your praent class laoder) and only after understanding the ClasLoader working I was able to resolve all the issues.
In My below blog piost I posted the solution as well
http://javaiscoool.blogspot.com/2012/12/deploy-jpa20-application-on-weblogic1033.html

December 24, 2012 at 9:10 PM

Riddhi said...
is there any change in working of ClassLoader on Java 5, 6 or Java 7 ? I also come to know that certain method on JDK executed by immediate classloader rather than delegation e.g. DriverManager.getConnection() and Class.forName(), is that true, if yes what issue they cause and what precaution Java programmer should take ?

February 12, 2013 at 10:22 PM

Anonymous said...
I have just few questions regarding class loaders :

When to create your own ClassLoader in Java?
What security consideration is required while implmenting ClassLoader in Java?
Can a untrusted code e.g. from Applet contains a ClasSLoader? Can it load classes from filesystem or network at same time?
What kind of ClassLoaders are bundled in JDK?
Can a ClassLoader load classes compiled in different Java version than class loader itself?

February 12, 2013 at 10:59 PM

Anonymous said...
Can we load our own class using bootstrap class loaders?

April 17, 2013 at 8:39 PM

jeet said...
As you said in the delegation principle "first request of loading this class will come to Application ClassLoader".If the order of loading classes is bootstrap loaders, then extension class loader then application class loader then why does the request first go to Application class loader.Should't it start from the bootstrap loader.Why is it so?

September 4, 2013 at 9:23 PM

govind said...
I have one query regarding this sentence
"By using separate ClassLoader you can also loads same class from multiple sources and they will be treated as different class in JVM"

This means there will be two version of same class and anyone of them will be picked up during runtime ?

March 23, 2014 at 11:54 PM

Dramebaaz said...
I am having same question as that of Govind, because somewhere i read that only once it will be loaded.
Also which part of memory loads the Java class ?

September 7, 2015 at 4:58 AM

Sunil kumar said...
To Govind & Dramebaaz :
By using separate class loader,it is possible to load same class from different sources.And those classes are associated to respective class loader only.
At runtime time,Each thread will be associated to only one class loader.So it will pick class based on which class loader is associated to your thread.
Note : Default class loader associated to thread is its parent thread class loader.

Suppose you have two class loaders which with parent child relation then it will search class from child class loader, if not exist then search in parent class loader and so on.

For suppose if a class loader loads same classes from different sources,then there are two versions of same class and any one of then can be picked at runtime.

October 15, 2015 at 7:57 AM

pantar said...
I did not understand belowline. Can you explain in detail

"Except bootstrap class loader all other class loaders are implemented using Java.lang.class loader".

Bootstrap class loader is implemented in c language. Where other class loaders implementations are provided in java.lang.class loader. This is what you mean to say.

December 13, 2015 at 5:50 AM

ZoomAll said...
Hi.

> 1) Bootstrap ClassLoader - JRE/lib/rt.jar

It not absolutely so. See result of:
System.getProperties().getProperty("sun.boot.class.path")

December 13, 2015 at 10:27 AM

ashutosh said...
Thread.currentThread().getContextClassLoader().loadClass(class_name) vs Class.forName() what is diff between two?

January 7, 2016 at 5:48 AM

javin paul said...
Hi @ashutosh, main difference between Thread.currentThread().getContextClassLoader().loadClass(class_name) vs Class.forName() method is that in case of first, context classloader, the classloader which has created the thread is used, while in second case the current classloader is used i.e. which loaded the class from which you are calling Class.forName() method.

January 8, 2016 at 8:20 PM

Anonymous said...
Thank you very very much for this nice article, now the entire class loader concept is clear. Thanks!!!!

February 25, 2016 at 10:29 AM

ashutosh said...
Thanks Javin..

February 26, 2016 at 6:01 AM

Unknown said...
It's Awsome Article on ClassLoaders Concept..

April 26, 2016 at 6:36 AM

Anonymous said...
"Can one class be loaded by two different ClassLoader in Java?" So, I guess the answer is YES?

Imagine we have a MammalClassLoader that has two child class loaders: CatClassLoader & DogClassLaoder, both are in the hierarchy below MammalClassLoader.

Now I let the CatClassLoader load the com.company.Food class which works fine as none of its parent class loaders has already loaded this class (= delegation check).

Now I want to load the same com.company.Food class using the DogClassLoader. This would also work as DogClassLoader checks if none of its parent class loaders has already loaded this class and indeed there are none as the CatClassLoader is not the parent of DogClassLoader. So DogClassLoader loads the same class com.company.Food again.

Still, com.company.Food (rather their instances) loaded by DogClassLoader & CatClassLoader would not pass the equality check as both are loaded by two different class loaders. So both Food classes are still unique. Still the same class was loaded twice!

Is the above reasoning correct?

August 13, 2016 at 4:22 AM

Anonymous said...
Great article with clear explanations! Thanks a lot!

September 6, 2016 at 7:21 PM

ASPOKN said...
Awesome article and diagram shows everything in on glance.

July 13, 2017 at 8:36 PM

Unknown said...
Is there any object created during loading of the class, if yes then what about static does it make sense of creation of object to static

August 26, 2019 at 12:43 PM

Post a Comment

Enter your comment...

Comment as: Google Accoun

Publish | Preview

Newer Post | Home | Older Post

Subscribe to: Post Comments (Atom)

Search This Blog

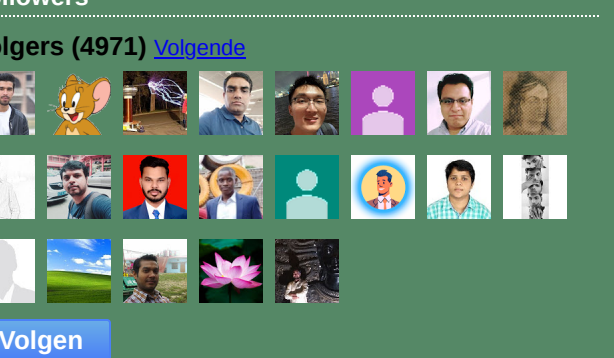
Search

Interview Questions
- core java interview question (174)
- data structure and algorithm (75)
- Coding Interview Question (74)
- interview questions (65)
- SQL Interview Questions (34)
- design patterns (33)
- object oriented programming (33)
- thread interview questions (30)
- collections interview questions (25)
- spring interview questions (23)
- database interview questions (16)
- servlet interview questions (15)
- Programming interview question (6)
- hibernate interview questions (6)

Best of Javarevisited
- How Spring MVC works internally?
- How to design a vending machine in Java?
- How HashMap works in Java?
- Why String is Immutable in Java?
- 10 Articles Every Programmer Must Read
- How to convert lambda expression to method reference in Java 8?
- 10 Tips to improve Programming Skill
- 10 OOP design principles programmer should know
- How Synchronization works in Java?
- 10 tips to work fast in Linux
- 5 Books to improve Coding Skills

Java Tutorials
- date and time tutorial (24)
- FIX protocol tutorial (15)
- Java Certification OCPJP SCJP (30)
- java collection tutorial (77)
- java IO tutorial (29)
- Java JSON tutorial (12)
- Java multithreading Tutorials (59)
- Java Programming Tutorials (19)
- Java xml tutorial (16)
- JDBC (33)
- jsp-servlet (37)
- online resources (216)

Followers
Volgers (4971) Volgende



Volgen

Translate This Blog
Select Language | Translate

Categories
- courses (244)
- SQL (65)
- database (46)
- linux (42)
- Eclipse (30)
- Java Certification OCPJP SCJP (30)
- JVM Internals (24)
- JQuery (21)
- REST (18)
- Maven (15)
- Testing (15)
- general (13)

Blog Archive
- ▼ 2021 (1008)
  - ► August (261)
  - ▼ July (470)
    - Parsing Large JSON Files using Jackson Streaming A...
    - How to Solve UnrecognizedPropertyException: Unreco...
    - How to parse JSON with date field in Java - Jackso...
    - Top 5 Advance Spring Boot + AWS + Azure Cloud Cour...
    - How to Ignore Unknown Properties While Parsing JSO...
    - Why use SerialVersionUID inside Serializable class...
    - How to Find Prime Factors of Integer Numbers in Ja...
    - java.lang.ClassNotFoundException: org.postgresql.D...
    - Why multiple inheritances are not supported in Java
    - How to create HTTP Server in Java - ServerSocket E...
    - Law of Demeter in Java - Principle of least Knowle...
    - How to do GROUP BY in Java 8? Collectors.groupingB...
    - 10 Things Every Java Programmer Should Know about ...
    - How to Find If Linked List Contains Loop or Cycle ...
    - 10 Tips to Debug Java Program in Eclipse - Examples
    - How SSL, HTTPS and Certificates Works in Java web ...
    - 3 Ways to Convert an Array to ArrayList in Java? E...
    - Difference between LEFT and RIGHT OUTER Joins in S...
    - Difference Between Linked List and Array in Java? ...
    - When to Make a Method Static in Java? Example
    - Different Types of JDBC Drivers in Java - Quick Ov...
    - Is it possible to have an abstract method in a fin...
    - Difference between ClassNotFoundException vs NoCla...
    - Top 5 Courses to learn Groovy and Grails in 2021 -...
    - Why Enum Singleton are better in Java? Examples
    - HashSet in Java – 10 Examples Programs Tutorial
    - Builder Design pattern in Java - Example Tutorial
    - 5 Coding Tips for Improving Performance of Java ap...
    - Difference between repaint and revalidate method i...
    - How to Count number of Set bits or 1's of Integer ...
    - When a class is loaded and initialized in JVM - Ja...
    - How to Add Two Integer Numbers without using Plus ...
    - Java ArrayList and HashMap Performance Improvement...
    - Is Swing Thread Safe in Java? Answer
    - Invalid initial and maximum heap size in JVM - How...
    - Top 20 jQuery Interview Questions and Answers
    - How to Close Java Program or Swing Application wit...
    - How to Check if Integer Number is Power of Two in ...
    - InvokeLater and InvokeAndWait in Java Swing (an ex...
    - How to Use Break, Continue, and Label in Loop in ...
    - How to get current URL, parameters and Hash tag us...
    - 10 Examples of HotSpot JVM Options in Java
    - Difference between Sun (Oracle) JVM and IBM JVM?
    - Top 30 JavaScript Interview Questions and Answers ...
    - How to Generate MD5 checksum for Files in Java? Ex...
    - How to find CPU and Memory used by Java process in...
    - 10 XSLT or XML, XSL Transformation Interview Quest...
    - How ClassLoader Works in Java? Example
    - 3 ways to solve java.lang.NoClassDefFoundError in ...
    - jQuery Tutorial - How to modify multiple HTML elem...
    - 20 Design Patterns and Software Design Interview Q...
    - How to use Comparator and Comparable in Java? With...
    - 10 Interview Questions on Java Generics for Progra...
    - What is -XX:+UseCompressedOops in 64 bit JVM? Example
    - Top 10 Garbage Collection Interview Questions and ...
    - How to Copy File in Java Program - Example Tutorial
    - What is Class File and Byte Code in Java? Example
    - 6 Best Free SQL Courses for Beginners to Learn in ...
    - Top 10 Java Swing Interview Questions Answers aske...
    - How to compare two lists of values in Microsoft Ex...
    - Difference between JVM, JIR, JRE, and JDK in Java?...
    - How to reload/refresh a page using JavaScript and ...
    - How to increase Heap memory of Apache Tomcat Serve...
    - How many characters allowed on VARCHAR(n) columns ...
    - What is bounded and unbounded wildcards in Generic...
    - Top 5 Courses to Learn Eclipse IDE for Java Progra...
    - How to Split String based on delimiter in Java? Ex...
    - Difference between Right shift and Unsigned right ...
    - What is the maximum Heap Size of 32 bit or 64-bit ...
    - How to Replace Line Breaks , New Lines From String...
    - How to Convert Byte Array to InputStream and Outpu...
    - How to Create JUnit Tests in Eclipse and NetBeans ...
    - 10 Articles Every Programmer Must Read
    - What is java.library.path? How to set in Eclipse I...
    - How to add and substract days in current date in J...
    - 10 JDK 7 Features to Revisit, Before You Welcome J...
    - Java Program to find factorial of number in Java -...
    - Top 50 Programmer Phone Interview Questions with A...
    - 7 Examples to Read File into a Byte Array in Java
    - Difference between Connected vs Disconnected RowSe...
    - Difference between Stub and Mock object in Java Un...
    - How to Add Leading Zeros to Integers in Java ? Str...
    - How to Implement Linked List in Java with JUnit Te...
    - Difference between FileInputStream and FileReader ...
    - Top 20 Amazon and Google Programming Interview Que...
    - Top 10 Puzzles, Riddles, Logical, and Lateral Thin...
    - Difference between UTF-8, UTF-16 and UTF-32 Charac...
    - How to Remove Given Character From String in Java ...
    - How to Implement Thread in Java with Example
    - Difference between valueOf and parseInt method in ...
    - How to Compare Two Enum in Java? Equals vs == vs C...
    - Difference Between Abstract Class vs Interface in ...
    - What is String args[] Argument in Java Main metho...
    - How to disable JUnit Test - @Ignore annotation Exa...
    - Java Program to Multiply Two Matrices - Matrix Mul...
    - The Ultimate Guide of Generics in Java - Examples
    - Difference between trunk, tags and branches in SVN...
    - How to Check If Number is Even or Odd without usin...
    - How to Convert InputStream to Byte Array in Java -...
    - Java Program to print Prime numbers in Java - Exa...
  - ► June (29)
  - ► May (35)
  - ► April (52)
  - ► March (55)
  - ► February (50)
  - ► January (56)
- ► 2020 (230)
- ► 2019 (44)
- ► 2018 (14)
- ► 2017 (142)
- ► 2016 (12)

References
1. Oracle's Java Tech Network
2. jQuery Documentation
3. Microsoft SQL Server Documentation
4. Java SE 8 API Documentation
5. Spring Documentation
6. Oracle's JAva Certification
7. Spring Security 5 Documentation

Pages
- Privacy Policy
- Terms and Conditions

Search This Blog

Search