Java Exception Interview Questions and Answers

APRIL 13, 2018 BY PANKAJ — 39 COMMENTS

Java provides a robust and object-oriented approach to handle exception scenarios known as **Java Exception Handling**.

Sometime back I wrote a long post on Exception Handling in Java and today I am listing some important **Java Exceptions Questions with Answers** to help you in interviews.

1. What is Exception in Java?

Exception is an error event that can happen during the execution of a program and disrupts it's normal flow. Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure etc.

Whenever any error occurs while executing a java statement, an exception object is created and then **JRE** tries to find exception handler to handle the exception. If suitable exception handler is found then the exception object is passed to the handler code to process the exception, known as **catching the exception**. If no handler is found then application throws the exception to runtime environment and JRE terminates the program.

Java Exception handling framework is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

2. What are the Exception Handling Keywords in Java?

There are four keywords used in java exception handling.

1. **throw**: Sometimes we explicitly want to create exception object and then throw it to halt the normal processing of the program. **throw** keyword is used to throw exception to the runtime to handle it.
2. **throws**: When we are throwing any checked exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it's caller method using `throws` keyword. We can provide multiple exceptions in the throws clause and it can be used with **main()** method also.
3. **try-catch**: We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
4. **finally**: finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurrs or not.

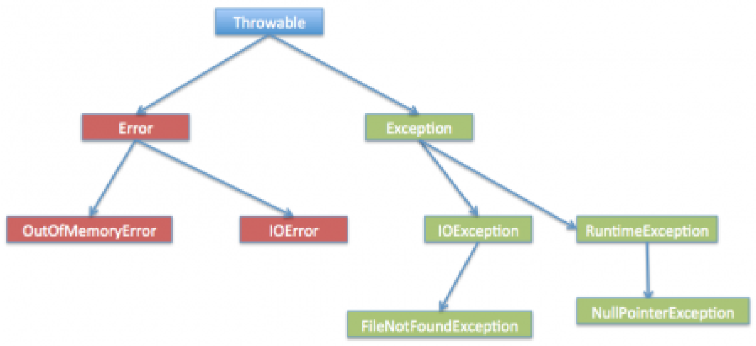
3. Explain Java Exception Hierarchy?

Java Exceptions are hierarchical and inheritance is used to categorize different types of exceptions. `Throwable` is the parent class of Java Exceptions Hierarchy and it has two child objects – `Error` and `Exception`. Exceptions are further divided into checked exceptions and runtime exception.

Errors are exceptional scenarios that are out of scope of application and it's not possible to anticipate and recover from them, for example hardware failure, JVM crash or out of memory error.

Checked Exceptions are exceptional scenarios that we can anticipate in a program and try to recover from it, for example FileNotFoundException. We should catch this exception and provide useful message to user and log it properly for debugging purpose. `Exception` is the parent class of all Checked Exceptions.

Runtime Exceptions are caused by bad programming, for example trying to retrieve an element from the Array. We should check the length of array first before trying to retrieve the element otherwise it might throw `ArrayIndexOutOfBoundException` at runtime. `RuntimeException` is the parent class of all runtime exceptions.



4. What are important methods of Java Exception Class?

Exception and all of it's subclasses doesn't provide any specific methods and all of the methods are defined in the base class Throwable.

1. **String getMessage()** – This method returns the message String of Throwable and the message can be provided while creating the exception through it's constructor.
2. **String getLocalizedMessage()** – This method is provided so that subclasses can override it to provide locale specific message to the calling program. Throwable class implementation of this method simply use `getMessage()` method to return the exception message.
3. **synchronized Throwable getCause()** – This method returns the cause of the exception or null id the cause is unknown.
4. **String toString()** – This method returns the information about Throwable in String format, the returned String contains the name of Throwable class and localized message.
5. **void printStackTrace()** – This method prints the stack trace information to the standard error stream, this method is overloaded and we can pass PrintStream or PrintWriter as argument to write the stack trace information to the file or stream.

5. Explain Java 7 ARM Feature and multi-catch block?

If you are catching a lot of exceptions in a single try block, you will notice that catch block code looks very ugly and mostly consists of redundant code to log the error, keeping this in mind Java 7 one of the feature was multi-catch block where we can catch multiple exceptions in a single catch block. The catch block with this feature looks like below:

```
catch(IOException | SQLException | Exception ex){
     logger.error(ex);
     throw new MyException(ex.getMessage());
}
```

Most of the time, we use finally block just to close the resources and sometimes we forget to close them and get runtime exceptions when the resources are exhausted. These exceptions are hard to debug and we might need to look into each place where we are using that type of resource to make sure we are closing it. So java 7 one of the improvement was **try-with-resources** where we can create a resource in the try statement itself and use it inside the try-catch block. When the execution comes out of try-catch block, runtime environment automatically close these resources. Sample of try-catch block with this improvement is:

```
try (MyResource mr = new MyResource()) {
            System.out.println("MyResource created in try-with-resources");
        } catch (Exception e) {
            e.printStackTrace();
        }
```

Read more about this at **Java 7 ARM**.

6. What is difference between Checked and Unchecked Exception in Java?

1. Checked Exceptions should be handled in the code using try-catch block or else method should use throws keyword to let the caller know about the checked exceptions that might be thrown from the method. Unchecked Exceptions are not required to be handled in the program or to mention them in throws clause of the method.
2. `Exception` is the super class of all checked exceptions whereas `RuntimeException` is the super class of all unchecked exceptions. Note that RuntimeException is the child class of Exception.
3. Checked exceptions are error scenarios that requires to be handled in the code, or else you will get compile time error. For example, if you use FileReader to read a file, it throws `FileNotFoundException` and we must catch it in the try-catch block or throw it again to the caller method. Unchecked exceptions are mostly caused by poor programming, for example NullPointerException when invoking a method on an object reference without making sure that it's not null. For example, I can write a method to remove all the vowels from the string. It's the caller responsibility to make sure not to pass null string. I might change the method to handle these scenarios but ideally the caller should take care of this.

7. What is difference between throw and throws keyword in Java?

throws keyword is used with method signature to declare the exceptions that the method might throw whereas throw keyword is used to disrupt the flow of program and handing over the exception object to runtime to handle it.

8. How to write custom exception in Java?

We can extend `Exception` class or any of it's subclasses to create our custom exception class. The custom exception class can have it's own variables and methods that we can use to pass error codes or other exception related information to the exception handler.

A simple example of custom exception is shown below.

```
package com.journaldev.exceptions;

import java.io.IOException;

public class MyException extends IOException {

        private static final long serialVersionUID = 4664456874499611218L;

        private String errorCode="Unknown_Exception";

        public MyException(String message, String errorCode){
                super(message);
                this.errorCode=errorCode;
        }

        public String getErrorCode(){
                return this.errorCode;
        }


}
```

9. What is OutOfMemoryError in Java?

OutOfMemoryError in Java is a subclass of java.lang.VirtualMachineError and it's thrown by JVM when it ran out of heap memory. We can fix this error by providing more memory to run the java application through java options.

```
$>java MyProgram -Xms1024m -Xmx1024m -XX:PermSize=64M -XX:MaxPermSize=256m
```

10. What are different scenarios causing "Exception in thread main"?

Some of the common main thread exception scenarios are:

- **Exception in thread main java.lang.UnsupportedClassVersionError**: This exception comes when your java class is compiled from another JDK version and you are trying to run it from another java version.
- **Exception in thread main java.lang.NoClassDefFoundError**: There are two variants of this exception. The first one is where you provide the class full name with .class extension. The second scenario is when Class is not found.
- **Exception in thread main java.lang.NoSuchMethodError: main**: This exception comes when you are trying to run a class that doesn't have main method.
- **Exception in thread "main" java.lang.ArithmeticException**: Whenever any exception is thrown from main method, it prints the exception is console. The first part explains that exception is thrown from main method, second part prints the exception class name and then after a colon, it prints the exception message.

Read more about these at Java Exception in Thread main.

11. What is difference between final, finally and finalize in Java?

final and finally are keywords in java whereas finalize is a method.

final keyword can be used with class variables so that they can't be reassigned, with class to avoid extending by classes and with methods to avoid overriding by subclasses, finally keyword is used with try-catch block to provide statements that will always gets executed even if some exception arises, usually finally is used to close resources. finalize() method is executed by Garbage Collector before the object is destroyed, it's great way to make sure all the global resources are closed.

Out of the three, only finally is related to java exception handling.

12. What happens when exception is thrown by main method?

When exception is thrown by main() method, Java Runtime terminates the program and print the exception message and stack trace in system console.

13. Can we have an empty catch block?

We can have an empty catch block but it's the example of worst programming. We should never have empty catch block because if the exception is caught by that block, we will have no information about the exception and it wil be a nightmare to debug it. There should be at least a logging statement to log the exception details in console or log files.

14. Provide some Java Exception Handling Best Practices?

Some of the best practices related to Java Exception Handling are:

- Use Specific Exceptions for ease of debugging.
- Throw Exceptions Early (Fail-Fast) in the program.
- Catch Exceptions late in the program, let the caller handle the exception.
- Use Java 7 ARM feature to make sure resources are closed or use finally block to close them properly.
- Always log exception messages for debugging purposes.
- Use multi-catch block for cleaner close.
- Use custom exceptions to throw single type of exception from your application API.
- Follow naming convention, always end with Exception.
- Document the Exceptions Thrown by a method using @throws in javadoc.
- Exceptions are costly, so throw it only when it makes sense. Else you can catch them and provide null or empty response.

Read more about them in detail at Java Exception Handling Best Practices.

15. What is the problem with below programs and how do we fix it?

In this section, we will look into some programming questions related to java exceptions.

1. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.FileNotFoundException;
import java.io.IOException;

public class TestException {

        public static void main(String[] args) {
                try {
                        testExceptions();
                } catch (FileNotFoundException | IOException e) {
                        e.printStackTrace();
                }
        }



        public static void testExceptions() throws IOException,
FileNotFoundException{
		}
```

Above program won't compile and you will get error message as "The exception FileNotFoundException is already caught by the alternative IOException". This is because FileNotFoundException is subclass of IOException, there are two ways to solve this problem.

First way is to use single catch block for both the exceptions.

```
		try {
			testExceptions();
		}catch(FileNotFoundException e){
			e.printStackTrace();
		}catch (IOException  e) {
			e.printStackTrace();
		}
```

Another way is to remove the FileNotFoundException from multi-catch block.

```
		try {
			testExceptions();
		}catch (IOException  e) {
			e.printStackTrace();
		}
```

You can chose any of these approach based on your catch block code.

2. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.FileNotFoundException;
import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException1 {

	public static void main(String[] args) {
			try {
				go();
			} catch (IOException e) {
				e.printStackTrace();
```

```
                        } catch (FileNotFoundException e) {
                                e.printStackTrace();
                        } catch (JAXBException e) {
                                e.printStackTrace();
                        }
        }

        public static void go() throws IOException, JAXBException,
```

The program won't compile because FileNotFoundException is subclass of IOException, so the catch block of FileNotFoundException is unreachable and you will get error message as "Unreachable catch block for FileNotFoundException. It is already handled by the catch block for IOException".

You need to fix the catch block order to solve this issue.

```
                        try {
                                go();
                        } catch (FileNotFoundException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                                e.printStackTrace();
                        } catch (JAXBException e) {
                                e.printStackTrace();
                        }
```

Notice that JAXBException is not related to IOException or FileNotFoundException and can be put anywhere in above catch block hierarchy.

3. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException2 {

        public static void main(String[] args) {
                try {
                        foo();
                } catch (IOException e) {
```

```
				e.printStackTrace();
			}catch(JAXBException e){
				e.printStackTrace();
			}catch(NullPointerException e){
				e.printStackTrace();
			}catch(Exception e){
				e.printStackTrace();
			}
		}
```

The program won't compile because JAXBException is a checked exception and foo() method should throw this exception to catch in the calling method. You will get error message as "Unreachable catch block for JAXBException. This exception is never thrown from the try statement body".

To solve this issue, you will have to remove the catch block of JAXBException.

Notice that catching NullPointerException is valid because it's an unchecked exception.

4. **What is the problem with below program?**

```
package com.journaldev.exceptions;

public class TestException3 {

	public static void main(String[] args) {
		try{
		bar();
		}catch(NullPointerException e){
			e.printStackTrace();
		}catch(Exception e){
			e.printStackTrace();
		}

		foo();
	}

	public static void bar(){

	}

	public static void foo() throws NullPointerException{
```

This is a trick question, there is no problem with the code and it will compile successfully. We can always catch Exception or any unchecked exception even if it's not in the throws clause of the method.

Similarly if a method (foo) declares unchecked exception in throws clause, it is not mandatory to handle that in the program.

5. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

public class TestException4 {

        public void start() throws IOException{
        }

        public void foo() throws NullPointerException{

        }
}

class TestException5 extends TestException4{

        public void start() throws Exception{
        }

        public void foo() throws RuntimeException{

        }
```

The above program won't compile because start() method signature is not same in subclass. To fix this issue, we can either change the method singnature in subclass to be exact same as superclass or we can remove throws clause from subclass method as shown below.

```
@Override
        public void start(){
        }
```

6. **What is the problem with below program?**

```
package com.journaldev.exceptions;

import java.io.IOException;

import javax.xml.bind.JAXBException;

public class TestException6 {

        public static void main(String[] args) {
                try {
                        foo();
                } catch (IOException | JAXBException e) {
                        e = new Exception("");
                        e.printStackTrace();
                }catch(Exception e){
                        e = new Exception("");
                        e.printStackTrace();
                }
        }

        public static void foo() throws IOException, JAXBException{
```

The above program won't compile because exception object in multi-catch block is final and we can't change it's value. You will get compile time error as "The parameter e of a multi-catch block cannot be assigned".

We have to remove the assignment of "e" to new exception object to solve this error.

Read more at Java 7 multi-catch block.

Thats all for java exception interview questions, I hope you will like it. I will be adding more to the list in future, make sure you bookmark it for future use.

« **PREVIOUS**
JSP Interview Questions and Answers

NEXT »
Eclipse log4j.xml – log4j.dtd cannot be validated as the XML definition

About Pankaj
If you have come this far, it means that you liked what you are reading. Why not reach little more and connect with me directly on **Google Plus**, **Facebook** or **Twitter**. I would love to hear your thoughts and opinions on my articles directly.

Recently I started creating video tutorials too, so do check out my videos on **Youtube**.

FILED UNDER: INTERVIEW QUESTIONS, JAVA

Comments

Jitendra Singh says
MAY 15, 2018 AT 10:54 AM

Beautiful tutorials.We love to study java from this tutorials all time!! Keep it Up!!
Reply

Shubham Negi says
APRIL 12, 2018 AT 2:12 PM
Hi,
Do you really think than question 6 has the right explanation. [Rest of the questions I haven't read]
This is totally wrong, How you are going to handle "checked exceptions in the try catch block".
Unchecked exceptions not meant to be handled using try-catch.?
Or it's some kind of typo mistake.
Reply

Pankaj says
APRIL 13, 2018 AT 2:10 AM
I didn't get your point, let's say you have a foo() method that is throwing FileNotFoundException and NullPointerException. Now I am calling foo() method in my program, since FileNotFoundException is a checked exception, I will have to either use try-catch block to handle it or use throws clause to throw it back to caller.
For NullPointerException, I don't have to do anything at all. I am not required to use catch block or add it to throws clause. However better programming would be to make sure that if I am passing any input to foo() method then make sure it's not null.
Reply

Anil says
JANUARY 9, 2018 AT 8:42 PM
Good explanation, must watch
https://www.youtube.com/watch?v=nUWd6242Myw
Reply

bharathi gowd says
FEBRUARY 21, 2017 AT 2:16 AM
this is the best way of student activity.super good but students not utilise this proper way
Reply

Kamet says
MARCH 10, 2016 AT 2:14 AM
Its written that "Exception is the super class of all checked exceptions whereas RuntimeException is the super class of all unchecked exceptions." According to hierarchy :-
Child of exception class are : IOException ,RuntimeException,ClassNotFoundException, CloneNotSupportedException.
So according to this "Exception is the super class of all checked exceptions" ,all the above are checked ??
Reply

Pankaj says
JUNE 28, 2016 AT 6:18 AM
Think of this like a Tree. Exception is the Root branch and then we have RuntimeException branch. Now anything coming out of RuntimeException branch will become unchecked exception. All others will be checked exceptions.
Exception is the superset. Anything extending RuntimeException is part of a child set and unchecked exceptions, if not then it's checked exception.
Reply

Veranga says
DECEMBER 10, 2015 AT 9:11 AM
Very important tutorial
Reply

Pankaj says
JUNE 28, 2016 AT 6:19 AM
Thanks a lot Veranga.
Reply

Urvee says
OCTOBER 7, 2015 AT 5:52 AM
Very informative and ease to understand !
However I have a point to add here – It is not mandatory to have catch after every try block, finally can be followed after a try block.
ie. Each try block must be followed by catch OR / AND finally.

(This was one of the tricky questions in some of the interviews I conducted and most of the candidates failed to answer correctly !)

Reply

Ravi Kumar says

APRIL 29, 2016 AT 4:27 PM

Each try block must be followed by catch OR finally.

Reply

Andreea Ciocan says

NOVEMBER 22, 2017 AT 2:30 AM

When we have try with resources we are allowed skip catch and finally blocks.

Reply

Pankaj says

JUNE 28, 2016 AT 6:20 AM

Yes you are right, you can have try-finally block too.

Reply

Ruchi Gupta says

DECEMBER 16, 2014 AT 11:26 PM

Hi Pankaj,
Could you please explain Progarm F, i couldnt understand it.

Reply

Pankaj says

JUNE 28, 2016 AT 6:22 AM

When you use multi-catch block like `catch (IOException | JAXBException e)`, the exception object implicitly becomes final. So you can't change it.
That's why you will get compilation error in `e = new Exception("");` statement in the catch block.
It's a very simple program, you can copy it in Eclipse and check for yourself.

Reply

Indresh says
JANUARY 30, 2018 AT 1:25 AM
same is applicable to try block resources, they are also final i.e. you cant re assign new value (object) to resources.
Reply

Rajeshwari says
NOVEMBER 20, 2014 AT 9:49 AM
Very informative.plz add few more questions
Reply

Pankaj says
JUNE 28, 2016 AT 6:23 AM
I will when I get more, but if you got some new and tricky questions then feel free to comment here.
Reply

prabhat says
NOVEMBER 10, 2014 AT 11:24 AM
can u re-explain throw and throws in detail ??
and also msg me the ans on my id please..
Reply

Pankaj says
JUNE 28, 2016 AT 6:25 AM
It's simple. 'throw' keyword is use to send exception to the caller program. 'throws' keyword is used in method declaration to let caller program know that the method may throw these xxx exceptions, so make sure you handle them.
Reply

Prathap says

AUGUST 2, 2014 AT 1:05 AM

my question is about QN:E, As per answer start() signature is different in subclass (which is creating the problem) and what about foo() which is also having same issue?

Reply

Aditya says

FEBRUARY 25, 2015 AT 10:10 PM

If a method in parent class throws checked exception, child class can not change the signature. The child can ignore it or add more unchecked exceptions.
If parent class method does not throw checked exception, the child class method cannot throw it either.

Reply

Pankaj says

JUNE 28, 2016 AT 6:27 AM

start() method is throwing checked exceptions, so the rule apply. foo() method is throwing un-checked exceptions and the rule doesn't apply on them.

Reply

XCESS says

JULY 31, 2014 AT 10:32 AM

Had great Help

Reply

sampath says

JULY 10, 2014 AT 8:59 AM

Looks like there is mistake in answer for question 11

final keyword can be used with class variables to make them immutable

final keyword can used for variables to make variables can't be re-assigned . it won't make variables immutable, immutable means when we can't change the object but we can re-assign a different object to same reference variable.

Reply

Pankaj says

JULY 10, 2014 AT 11:32 AM

Yes you are right, updated the post. Thanks!!

Reply

Manjunatha Kuruba says

JUNE 29, 2014 AT 6:03 AM

try {
foo();
} catch (IOException | JAXBException e) {
e = new Exception("");
e.printStackTrace();
}catch(Exception e){
e = new Exception("");
e.printStackTrace();
}
only the exception object for IOException | JAXBException exceptions is final but exception object for "Exception" exception is not final.

Reply

Pankaj says

JUNE 29, 2014 AT 10:14 PM

Yes, thats what I meant with multi-catch block.

Reply

Manjunath says

MAY 28, 2014 AT 3:05 AM

Great man… such a descriptive way of explanation..
Thanks

Reply

Kalai says

MAY 17, 2014 AT 10:21 PM

Hats off to you Pankaj ..Super ..good work….Never stop your blogging habbit…

Reply

sonal says

MAY 9, 2014 AT 10:31 AM

Great post "Hats Off To YOU"…. I have read many offline/online books n tutorials but ur understanding to subject is so deep that it reflects in ur posts….not only this topic but every of ur post …. exceptional knowledge and exceptional writing……

Regards

Sonal

Reply

Pankaj says

MAY 10, 2014 AT 12:03 AM

Thanks Sonal, your appreciation means a lot.

Reply

Sachidananda Dash says

APRIL 2, 2014 AT 5:28 PM

finally() method is executed by Garbage Collector before the object is destroyed is probably wrong , Ideally it should be finalize()

Reply

Pankaj says

APRIL 2, 2014 AT 7:29 PM

Yes you are right, it was a typo. Thanks for pointing out, I have updated the post.

Reply

Rajiv Rahul says

DECEMBER 14, 2013 AT 7:02 AM

Hi,

Thank you so much for the great work that you are doing for us which is benefiting us in building our career. In fact Its our duty to appreciate people like you because you deserve it!

Reply

Pankaj says

JUNE 28, 2016 AT 6:34 AM

Thanks for the nice comments Rajiv.
Reply

Sidram says
SEPTEMBER 23, 2013 AT 6:02 AM
It very nice explanation. I have one qns, what is the difference between Exception & RuntimeException. When i create custom exception by extedning RuntimeException which is becomes unchecked exception, How it can be when RuntimeException allready extends Exception class. How it will be differentites. Could you explain.
Reply

Irshad says
OCTOBER 28, 2013 AT 7:10 PM
There is a little bit change in answer of Question E is
To fix this issue, we can either change the method signature in subclass to be exact same as superclass or we can remove throws clause from subclass or can define the subclass of Exception in child class method as shown below.
@Override
public void start() throws FileNotFoundException{
}
Reply

zakir says
JULY 13, 2014 AT 6:02 AM
Good one…very informative post
Reply

Leave a Reply

Your email address will not be published. Required fields are marked *
Comment

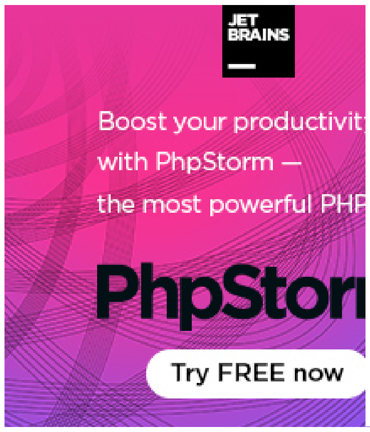
Name *

Email *

Save my name, email, and website in this browser for the next time I comment.

POST COMMENT

Search for tutorials...



DOWNLOAD ANDROID APP

GET IT ON Google Play

CORE JAVA TUTORIAL

Java 10 Tutorials Java 9 Tutorials Java 8 Tutorials Java 7 Tutorials Core Java Basics OOPS Concepts Data Types and Operators String Manipulation Java Arrays Annotation and Enum Java Collections Java IO Operations Java Exception Handling MultiThreading and Concurrency Regular Expressions Advanced Java Concepts

IMPORTANT INTERVIEW QUESTIONS

Java Interview Questions

- Core Java Interview Questions
- String Interview Questions
- Multithreading Interview Questions
- Collections Interview Questions
- Exception Interview Questions
- Java Programming Interview Questions
- Java 8 Interview Questions Part 1
- Java 8 Interview Questions Part 2
- Servlet Interview Questions
- JSP Interview Questions
- Struts 2 Interview Questions
- JDBC Interview Questions
- Spring Interview Questions
- Hibernate Interview Questions
- JSF Interview Questions
- Web Services Interview Questions
- Scala Basic Interview Questions
- Scala Intermediate Interview Questions
- Scala Advanced Interview Questions
- Scala Interview Questions Summary
- Common Job Interview Questions

Miscellaneous

- Java ClassLoader
- String StringBuffer StringBuilder
- Java is Pass By Value
- Java Heap vs Stack Memory

RECOMMENDED TUTORIALS

Java Tutorials

- Java IO
- Java Regular Expressions
- Multithreading in Java
- Java Logging
- Java Annotations
- Java XML
- Collections in Java
- Java Generics
- Exception Handling in Java
- Java Reflection
- Java Design Patterns
- JDBC Tutorial

Java EE Tutorials

- Servlet JSP Tutorial
- Struts2 Tutorial
- Spring Tutorial
- Hibernate Tutorial
- Primefaces Tutorial
- Apache Axis 2
- JAX-RS
- Memcached Tutorial

Learn Java

Java Code

Java Exception Error

Enable Java

Java Problems

Java Classes

Java Language

Java Virtual Machine Download