**CoreJava Interview Queations**

ackage is also a collection of classes interfaces and sub sub packages etc.

**Why Java take 2 byte of memory for store character ?**

Java support more than 18 international languages so java take 2 byte for characters, because for 18 international language 1 byte of memory is not sufficient for storing all characters and symbols present in 18 languages. Java supports Unicode but c support ascii code. In ascii code only English language are present, so for storing all English latter and symbols 1 byte is sufficient.

**What is java and javac ?**

Java and javac are tools or application programs or exe files developed by sun micro system and supply as a part of jdk 1.5/1.6/1.7/1.8 in bin folder. Java tool are used for run the java program and javac tool are used for compile the java program.

**Why Using naming Conversion ?**

Different Java programmers can have different styles and approaches to write program. By using standard Java naming conventions they make their code easier to read for themselves and for other programmers. Readability of Java code is important because it means less time is spent trying to figure out what the code does, and leaving more time to fix or modify it.

**Which syntax follow by java for naming conversion ?**

Java follows camelcase syntax for naming the class, interface, method and variable.

[Read in Detail about naming conventions........](http://www.tutorial4us.com/java/Naming-Conversion)

**What is JVM ?**

JVM (Java Virtual Machine) is a software. It is a specification that provides runtime environment in which java bytecode can be executed.

[Read in Detail about JVM........](http://www.tutorial4us.com/java/JVM-Architecture)

**What is operation of JVM ?**

JVM mainly performs following operations.

* Allocating sufficient memory space for the class properties.
* Provides runtime environment in which java bytecode can be executed
* Converting byte code instruction into machine level instruction.

**What is classloader ?**

Class loader subsystem will load the **.class** file into java stack and later sufficient memory will be allocated for all the properties of the java program into following five memory locations.

* Heap area
* Method area
* Java stack
* PC register
* Native stack

**What is Main aim of JIT Compiler ?**

The main aim of JIT compiler is to speed up the execution of java program.

**Why Boolean data types take zero byte memory ?**

Boolean data type takes zero bytes of main memory space because Boolean data type of java implemented by Sun Micro System with a concept of flip - flop. A flip - flop is a general purpose register which stores one bit of information (one true and zero false).

**Why Java is Simple and easy**

Java is simple because of the following factors:

* Java is free from pointer due to this execution time of application is improve. [whenever we write a Java program we write without pointers and internally it is converted into the equivalent pointer program].
* Java have Rich set of API (application protocol interface).
* Java have garbage collector which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.
* Java contains user friendly syntax for developing Java applications.

**How Java have high performance ?**

Java have high performance because;

* Java is use Bytecode which is more faster than ordinary pointer code so Performance of java is high.
* Garbage collector, collect the unused memory space and improve the performance of java application.
* Java have no pointers so that using java program we can develop an application very easily.
* It support multithreading, because of this time consuming process can be reduced to execute the program.

**How Java is Platform Independent**

A programming language or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents Operating System).

Java is a platform independent programming language, Because when you install jdk software on your system then automatically jvm are install on your system. For every operating system separate jvm is available which is capable to read **.class** file or **byte code**. When we compile your java code then .class file is generated by javac compiler these code are readable by jvm and every operating system have its own jvm so jvm is platform dependent but due to jvm java language is become platform independent.

**Difference between conditional and looping statement**

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

**Why a static block executes before the main method ?**

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

**Can we override static method ?**

No, static method cannot be overridden.

**Why we cannot override static method ?**

because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

**Which Java operator is right associative ?**

The = operator is right associative.

**What is dot operator ?**

The dot operator(.) is used to access the instance variables and methods of class objects.It is also used to access classes and sub-packages from a package.

**Why use this keyword**

The main purpose of using this keyword is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

**When Need of super keyword ?**

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

**What is the difference between this. (this dot) and this() (this off).**

this. can be used to differentiate variable of class and formal parameters of method or constructor.

this() can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

**Difference between static and final keyword**

static keyword always fixed the memory that means that will be located only once in the program where as final keyword always fixed the value that means it makes variable values constant.

**why main method is static ?**

Because object is not required to call static method if main() is non-static method, then jvm create object first then call main() method due to that face the problem of extra memory allocation.

**We can overload main() method ?**

Yes, We can overload main() method. A Java class can have any number of main() methods. But run the java program, which class should have main() method with signature as **"public static void main(String[] args)**. If you do any modification to this signature, compilation will be successful. But, not run the java program. we will get the run time error as main method not found.

**Can we override java main method ?**

No, because main is a static method.

**Difference between Path and ClassPath**

Path variable is set for use all the tools like java, javac, javap, javah, jar, appletviewer etc.

Classpath variable is used for set the path for all classes which is used in our program so we set classpath upto rj.jar. in rt.jar file all the .class files are present. When we decompressed rt.jar file we get all .class files.

**What do you mean by portable ?**

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, Pascal are treated as non-portable language. JAVA is a portable language.

**Why use Import keyword in java ?**

Import is a keyword in java language used to import the predefined properties of java API into current working java program. [Read more......](http://www.tutorial4us.com/java/Import-Statement)

**What is Wrapper Classes ?**

For each and every fundamental data type there exist a pre-defined class, Such predefined class is known as wrapper class.

**Why use Wrapper Classes ?**

The main purpose of wrapper class is to convert numeric string data into numerical or fundamental data. We know that in java whenever we get input form user, it is in the form of string value so here we need to convert these string values in different different datatype (numerical or fundamental data), for this conversion we use wrapper classes. [Read more......](http://www.tutorial4us.com/java/Wrapper-Classes)

**Access Specifier**

**Which access specifier are called universal access specifier ?**

Public

**Which access specifier is not a keyword ?**

Default

**Which access specifier is package level access specifier ?**

Default

**Scope of protected specifier ?**

protected members of class is accessible within the same class and other class of same package and also accessible in inherited class of other package.

**Scope of private access specifier ?**

private members of class in not accessible any where in program these are only accessible within the class. private are also called class level access specifier.

Write a program in java which prints the numbers from 1 to 100. But, multiples of 3 should be replaced with "Fizz", multiples of 5 should be replaced with "Buzz" and multiples of both 3 and 5 should be replaced with "FizzBuzz"?.

**Example Fizz Buzz Problem**

**publicclassFizzBuzzProblem**

**{**

**publicstaticvoid main(String args[])**

**{**

**for(int i = 1; i <= 100; i++)**

**{**

**if((i % (3\*5)) == 0)**

**{**

**System.out.println("FizzBuzz");**

**}**

**elseif ((i % 5) == 0)**

**{**

**System.out.println("Buzz");**

**}**

**elseif ((i % 3) == 0)**

**{**

**System.out.println("Fizz");**

**}**

**else**

**{**

**System.out.println(i);**

**}**

**}**

**}**

**}**

**Constructor**

**Why use constructor ?**

The main purpose of create a constructor is, for placing user defined values in place of default values.

**Why constructor not return any value ?**

Constructor will never return any value even void, because the basic aim constructor is to place value in the object

**Why constructor definition should not be static ?**

Constructor definition should not be static because constructor will be called each and every time when object is created. If you made constructor is static then the constructor before object creation same like main method.

**Why constructor is not inherited ?**

Constructor will not be inherited from one class to another class because every class constructor is created for initialize its own data members.

**What is purpose of default constructor ?**

The purpose of default constructor is to create multiple object with respect to same class for placing same value.

**What is purpose of parameterized constructor ?**

The purpose of parametrized constructor is to create multiple object with respect to same class for placing different value of same type or different type or both.

**Is constructor inherited?**

No, constructor is not inherited.

**Can you make a constructor final?**

No, constructor can't be final.

**What is the purpose of default constructor?**

The default constructor provides the default values to the objects. The java compiler creates a default constructor only if there is no constructor in the class.

**Does constructor return any value?**

yes, that is current instance (You cannot use return type yet it returns a value).

**What is flow of constructor in Java ?**

Constructor are calling from bottom to top and executing from top to bottom.

**Why overriding is not possible at constructor level. ?**

The scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

**Difference between Method and Constructor**

|  |  |  |
| --- | --- | --- |
|  | **Method** | **Constructor** |
| 1 | Method can be any user defined name | Constructor must be class name |
| 2 | Method should have return type | It should not have any return type (even void) |
| 3 | Method should be called explicitly either with object reference or class reference | It will be called automatically whenever object is created |
| 1 | Method is not provided by compiler in any case. | The java compiler provides a default constructor if we do not have any constructor. |

**String Handling**

**Why use string handling in Java**

The basic aim of String Handling concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. String Handling provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

**Difference between String and StringBuffer**

**What is difference between equal() and == ?**

equals() method always used to comparing contents of both source and destination String. It return true if both string are same in meaning and case otherwise it returns false.

== Operator is always used for comparing references of both source and destination objects but not their contents.

|  |  |  |
| --- | --- | --- |
|  | **String** | **StringBuffer** |
| 1 | The data which enclosed within double quote (" ") is by default treated as String class. | The data which enclosed within double quote (" ") is not by default treated as StringBuffer class |
| 2 | String class object is immutable | StringBuffer class object is mutable |
| 3 | When we create an object of String class by default no additional character memory space is created. | When we create an object of StringBuffer class by default we get 16 additional character memory space. |

**When we use String, StringBuffer and StringBuilder**

* If the content is fixed and would not change frequently then we use String.
* If content is not fixed and keep on changing but thread safety is required then we use StringBuffer
* If content is not fixed and keep on changing and thread safety is not required then we use StringBuilder

**What is Similarities between String and StringBuffer**

* Both of them are belongs to public final. so that they never participates in inheritance that is is-A relationship is not possible but they can always participates in As-A and Uses-A relationship.
* We can not override the method of String and StringBuffer.

**Difference between StringBuffer and StringBuilder**

All the things between StringBuffer and StringBuilder are same only difference is StringBuffer is synchronized and StringBuilder is not synchronized. synchronized means one thread is allow at a time so it thread safe. Not synchronized means multiple threads are allow at a time so it not thread safe.

|  |  |  |
| --- | --- | --- |
|  | **StringBuffer** | **StringBuilder** |
| 1 | It is thread safe. | It is not thread safe. |
| 2 | Its methods are synchronized and provide thread safety. | Its methods are not synchronized and unable to provide thread safety. |
| 3 | Relatively performance is low because thread need to wait until previous process is complete. | Relatively performance is high because no need to wait any thread it allows multiple thread at a time. |
| 1 | Introduced in 1.0 version. | Introduced in 1.5 version. |

**What is StringTokenizer ?**

It is a pre defined class in java.util package can be used to split the given string into tokens (parts) based on delimiters (any special symbols or spaces). [Read more......](http://www.tutorial4us.com/java/StringTokenizer)

**Exception Handling**

**What is Exception Handling ?**

The process of converting system error messages into user friendly error message is known as Exception handling.

**What is Exception ?**

An exception is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

**Which is super class for any Exception class ?**

Object class is super class for any Exception class.

**Can any statement is possible between try and catch block ?**

Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

**Can any try block contain another try block ?**

Yes, One try block can contains another try block that is nested or inner try block can be possible.

**When IOException is thrown ?**

IOException is thrown in following conditions which is given below;

* Try to transfer more data but less data are present.
* Try to read data which is corrupted
* Try to write but file is read only.

**When ArithmeticException is thrown ?**

The ArithmeticException is thrown when integer is divided by zero or taking the remainder of a number by zero. It is never thrown in floating-point operations.

**Difference between throw and throws ?**

|  |  |  |
| --- | --- | --- |
|  | **throw** | **throws** |
| 1 | throw is a keyword used for hitting and generating the exception which are occurring as a part of method body | throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages |
| 2 | The place of using throw keyword is always as a part of method body. | The place of using throws is a keyword is always as a part of method heading |
| 3 | When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading | When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body. |

**Difference between checked Exception and un-checked Exception ?**

|  |  |  |
| --- | --- | --- |
|  | **Checked Exception** | **Un-Checked Exception** |
| 1 | checked Exception are checked at compile time | un-checked Exception are checked at run time |
| 2 | for checked Exception Extend Throwable class except RuntimeException. | for un-checked Exception extend RuntimeException. |
| 3 | e.g.  IOException, SQLException, FileNotFoundException etc. | e.g.  ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, NumberNotFoundException etc. |

**Difference between Error and Exception**

|  |  |  |
| --- | --- | --- |
|  | **Error** | **Exception** |
| 1 | Can be handle. | Can not be handle. |
| 2 | Example: NoSuchMethodError OutOfMemoryError | Example: ClassNotFoundException NumberFormateException |

**Multithreading**

**What is thread ?**

Thread is a lightweight components and it is a flow of control. In other words a flow of control is known as thread.

**What is multithreading ?**

Multithreading in java is a process of executing multiple threads simultaneously.

**Explaing State or Life cycle of thread.**

State of a thread are classified into five types they are

* New State
* Ready State
* Running State
* Waiting State
* Halted or dead State

[Read more.........](http://www.tutorial4us.com/java/java-multithreading)

**How to achieve multithreading in java ?**

In java language multithreading can be achieve in two different ways.

* Using thread class
* Using Runnable interface

**In which state no memory is available for thread ?**

If the thread is in new or dead state no memory is available but sufficient memory is available if that is in ready or running or waiting state.

**Difference between sleep() and suspend() ?**

Sleep() can be used to convert running state to waiting state and automatically thread convert from waiting state to running state once the given time period is completed. Where as suspend() can be used to convert running state thread to waiting state but it will never return back to running state automatically.

**What is Thread Synchronization ?**

Allowing only one thread at a time to utilized the same resource out of multiple threads is known as thread synchronization or thread safe.

**Why use Thread Synchronization ?**

Whenever multiple threads are trying to use same resource than they may be chance to of getting wrong output, to overcome this problem thread synchronization can be used.

**How to achieve Thread Synchronization in java ?**

In java language thread synchronization can be achieve in two different ways.

* Synchronized block
* Synchronized method

**How to achieve multiple inheritance in java ?**

Using Interface concept you can achieve multiple inheritance in java.

## Final, Static, This

### Why a static block executes before the main method ?

A class has to be loaded in main memory before we start using it. Static block is executed during class loading. This is the reason why a static block executes before the main method.

### Can we override static method ?

No, static method cannot be overridden.

### Why we cannot override static method ?

because static method is bound with class where as instance method is bound with object. Static belongs to class area and instance belongs to heap area.

### Why use this keyword

The main purpose of using this keyword is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

### When Need of super keyword ?

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

### What is the difference between this. (this dot) and this() (this off).

this. can be used to differentiate variable of class and formal parameters of method or constructor.

this() can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

### Difference between static and final keyword

static keyword always fixed the memory that means that will be located only once in the program where as final keyword always fixed the value that means it makes variable values constant.

### why main method is static ?

Because object is not required to call static method if main() is non-static method, then jvm create object first then call main() method due to that face the problem of extra memory allocation.

### Difference between non-static and static variable ?

|  |  |  |
| --- | --- | --- |
|  | **Non-Static method** | **Static method** |
| 1 | These method never be preceded by static keyword Example:  **void fun1()**  **{**  **......**  **......**  **}** | These method always preceded by static keyword Example:  **staticvoid fun2()**  **{**  **......**  **......**  **}** |
| 2 | Memory is allocated multiple time whenever method is calling. | Memory is allocated only once at the time of loading. |
| 3 | It is specific to an object so that these are also known as instance method. | These are common to every object so that it is also known as member method or class method. |
| 4 | These methods always access with object reference Syntax:  **Objref.methodname();** | These property always access with class reference Syntax:  **className.methodname();** |
| 5 | If any method wants to be execute multiple time that can be declare as non static. | If any method wants to be execute only once in the program that can be declare as static . |

### What is difference between super and this keyword

Super keyword is always pointing to base class features and this keyword is always pointing to current class features. [Read more.......](http://www.tutorial4us.com/java/java-this-keyword)

### What is difference between super(), super(..), this() and this(..).

super() and super(..) are used for establishing the communication between base class and derived class constructor.

this() and this(...) are used for establishing the communication between current class constructor.

### Which access specifiers is known as package access specifiers.

default access specifiers is known as package access specifiers.

### Why abstract class not made as final ?

Abstract classes definitions should not be made as final because abstract classes always participate in inheritance classes.

### Difference between non-static and static variable ?

|  |  |  |
| --- | --- | --- |
|  | **Non-Static method** | **Static method** |
| 1 | These method never be preceded by static keyword Example:  **void fun1()**  **{**  **......**  **......**  **}** | These method always preceded by static keyword Example:  **staticvoid fun2()**  **{**  **......**  **......**  **}** |
| 2 | Memory is allocated multiple time whenever method is calling. | Memory is allocated only once at the time of loading. |
| 3 | It is specific to an object so that these are also known as instance method. | These are common to every object so that it is also known as member method or class method. |
| 4 | These methods always access with object reference Syntax:  **Objref.methodname();** | These property always access with class reference Syntax:  **className.methodname();** |
| 5 | If any method wants to be execute multiple time that can be declare as non static. | If any method wants to be execute only once in the program that can be declare as static . |

### What is new keyword ?

A new keyword is used to allocate memory at runtime, new keyword is used for create an object of class

### When use volatile keyword ?

If the variable keep on changing such type of variables we have to declare with volatile modifier. [Read more......](http://www.tutorial4us.com/java/Volatile-Keyword)

### Main advantage of volatile keyword ?

The main advantage of Volatile keyword is we can resolve data inconsistency problems.

### Main dis-advantage of Volatile ?

The main dis-advantage of Volatile keyword is, crating and maintaining a separate copy for every thread, increases complexity of the programming and effects performance of the system.

### Why use synchronized keyword ?

Synchronized Keyword is used for when we want to allowed only one thread at a time then use Synchronized modifier. If a method or block declared as a Synchronized then at a time only one thread is allowed to operate on the given object.

### Main advantage of Synchronized keyword ?

The main advantage of Synchronized keyword is we can resolve data inconsistency problem.

### The main dis-advantage of Synchronized keyword ?

The main dis-advantage of Synchronized keyword is it increased the waiting time of thread and effect performance of the system, Hence if there is no specific requirement it is never recommended to use synchronized keyword.

# [5 ways to check if String is empty in Java - examples](http://javarevisited.blogspot.in/2013/02/5-ways-to-check-if-string-is-empty-in-java-examples.html)

String in Java is considered empty if its not null and it’s length is zero. By the way before checking length you should verify that String is not null because calling length() method on null String will result in [java.lang.NullPointerException](http://javarevisited.blogspot.sg/2012/06/common-cause-of-javalangnullpointerexce.html). Empty String is represented by String literal “”. Definition of empty String may be extended to those String as well which only contains white space but its an specific requirement and in general String with white space are not considered as empty String in Java. Since [String](http://javarevisited.blogspot.com/2011/07/string-vs-stringbuffer-vs-stringbuilder.html) is one of the most frequently used class and commonly used in method arguments, we often needs to check if String is empty or not. Thankfully there are multiple ways to find *if String is empty in Java or not*. You can also count number of characters in String, as [String is represented as character array](http://javarevisited.blogspot.com/2011/10/how-substring-in-java-works.html) and decide if String is empty or not. If count of characters is zero than its an empty String. In this Java String tutorial we going to see 5 ways to find if any String in Java is empty or not. Here are our five ways to check empty String :

1) Checking if String is empty by using String.length()

2) Find  if String is empty by using [equals() method](http://javarevisited.blogspot.sg/2011/02/how-to-write-equals-method-in-java.html) of String

3) Checking if String is empty by using isEmpty() method String, only available from Java 6 onwards.

4) Find if String is empty using Apache commons StringUtils class

5) Using Spring framework’s StringUtils.hasLength() method.

**Find if String is empty by checking length**

It's the most easy and popular method to verify if String is empty or not. You can find length of String by calling

length() method which actually returns number of characters in String. Be careful to check if String is null before calling length()to avoid [NullPointerException](http://java67.blogspot.com/2012/09/what-is-nullpointerexception-in-java.html). here is an example to *check is String empty using length*:

if(string != **null** && string.length() == 0){ **return** **true**; }

**String empty using equals method**

You can also [compare String](http://javarevisited.blogspot.com/2012/03/how-to-compare-two-string-in-java.html) to empty String literal "" to check if it’s empty or not. [equals method in Java](http://javarevisited.blogspot.sg/2012/12/difference-between-equals-method-and-equality-operator-java.html) returns false if other argument is null, so it automatically checks for null string as well. Here is code example of checking emptiness of String using equals:

**public** **static** **boolean** isStringEmptyByEquals(**String** input){  
        **return** "".equals(input);  
}

**Use isEmpty() method of Java 6**

You can also check if String is empty or not by using isEmpty() method of [String class](http://javarevisited.blogspot.sg/2012/03/how-to-compare-two-string-in-java.html) added in Java6. This is by far most readable way but *you need to check if String is null* or not before calling isEmpty() method. see code example section for use of isEmpty() method.

**String Empty check using Apache Commons lang StringUtils**

Apache commons lang has a StringUtils class which has [static utility method](http://javarevisited.blogspot.com/2011/11/static-keyword-method-variable-java.html) isEmpty(String input), which returns true if input string is null or has length greater than zero. Note this is different than our first method because it consider null as empty String while we are here only considering zero length String as empty String.  If you are already using Apache commons lang in your project e.g. for [overriding toString method](http://javarevisited.blogspot.com/2012/09/override-tostring-method-java-tips-example-code.html), than you can use StringUtils instead of writing your own method. Check example section to see how to use StringUtils.isEmpty(), by the way here is output of StringUtils for some common input :

StringUtils.isEmpty("")        = true

StringUtils.isEmpty(null)      = true

StringUtils.isEmpty(" ")       = false

StringUtils.isEmpty("fix")     = false

StringUtils.isEmpty("  fix  ") = false

**Check if String is Empty in Java - Using Spring**

[Spring](http://javarevisited.blogspot.sg/2012/08/convert-collection-to-string-in-java.html) is popular Java framework and most of new projects uses Spring to take benefit of [dependency Injection](http://javarevisited.blogspot.com/2012/12/inversion-of-control-dependency-injection-design-pattern-spring-example-tutorial.html), it provide StringUtils class for performing common String operation. StringUtils provides method called hasLength(input String), which **returns true if String is not null and contains any character**, including white space. You can also use hasLength to determine if String is empty in Java or not. Next section has code examples of all five methods of checking empty string mentioned in this Java tutorial, by the way here is how hasLength() treats null and empty String :

StringUtils.hasLength("") = false

StringUtils.hasLength(null) = false

StringUtils.hasLength(" ") = true

StringUtils.hasLength("Hello") = true

## Code Example to verify if String is empty in Java

[](http://2.bp.blogspot.com/-jlIT7z2RuQg/T9whph8c-FI/AAAAAAAAAYw/TrAvT017-1c/s1600/spring_thumbnail.PNG)Here is complete code example of How to check if String is empty in Java. This program combines all approaches we have discussed so fart to check if Java String is empty or not. One interesting thing to note in this program is How I have used StringUtils from Spring Framework. Since there are two classes with same name but from different [package](http://java67.blogspot.sg/2012/08/what-is-package-in-java-how-to-use.html), i.e. StringUtils from Apache and Spring. You can only import one and you need to use other with its fully qualified name to avoid ambiguity.

**import** org.apache.commons.lang.StringUtils;  
  
**public** **class** StringEmptyTest {  
  
    **public** **static** **void** main(**String** args[]) {  
    
        **String** input1 = "";  
        **String** input2 = **null**;  
        **String** input3 ="abc";  
        
        *//determine if String is empty using length method , also checks if string is null*  
        **System**.out.println("checking if String empty using length");  
        **System**.out.println("String " + input1 + " is empty :" +isStringEmpty(input1) );  
        **System**.out.println("String " + input2 + " is empty :" +isStringEmpty(input2) );  
        **System**.out.println("String " + input3 + " is empty :" +isStringEmpty(input3) );  
        
        *//determine if String is empty using equals method*  
        **System**.out.println("find if String empty using equals");  
        **System**.out.println("String " + input2 + " is empty :" +isStringEmptyByEquals(input2) );  
        
         *//determine if String is empty using isEmpty of Java 6*  
        **System**.out.println("find if String empty using isEmpty method of Java 6");  
        **System**.out.println("String " + input3 + " is empty :" + input3.isEmpty());  
        
        *//determine if String is empty by Apache commons StringUtils*  
        **System**.out.println("check if String empty by commons StringUtils");  
        **System**.out.println("String " + input2 + " is empty :" + StringUtils.isEmpty(input2));  
        
        *//determine if String is empty by Spring framework StringUtils hasLength method*  
        **System**.out.println("check if String empty by Spring framework StringUtils");  
        **System**.out.println("String " + input2 + " is empty :" + org.springframework.util.StringUtils.hasLength(input2));  
  
        
    }  
    
    **public** **static** **boolean** isStringEmpty(**String** input){  
        if(input != **null** && input.length() == 0){  
            **return** **true**;  
        }  
        **return** **false**;  
    }  
    
    **public** **static** **boolean** isStringEmptyByEquals(**String** input){  
        **return** "".equals(input);  
    }  
}  
  
**Output:**  
checking if **String** empty using length  
**String**  is empty :**true**  
**String** **null** is empty :**false**  
**String** abc is empty :**false**

find if **String** empty using equals  
**String** **null** is empty :**false**

find if **String** empty using isEmpty method of Java 6  
**String** abc is empty :**false**

check if **String** empty by commons StringUtils  
**String** **null** is empty :**true**

check if **String** empty by **Spring** framework StringUtils  
**String** **null** is empty :**false**

That’s all on **How to check if String is empty in Java**. I thing Java 6 isEmpty() method is more readable than any other option but it’s not null safe which means either write your own method or use hasLength() from Spring Framework.  By the way be careful with null String as some programmer consider null string as empty String and even Apache commons StringUtils.isEmpty() method return true for null String.

**Question 1. What is exception in java?**

**Answer**. Java Exception handling provides a mechanism to handle compile and runtime errors.

* To make application robust Exception must be handled appropriately,
* by handling exceptions we end up giving some meaningful message to end user rather than giving meaningless message.

**Question 2. Explain exception hierarchy in java?**

**Answer**.

**Exception hierarchy >**

java.lang.**Object** is superclass of all classes in java.

java.lang.**Throwable** is superclass of java.lang.**Exception** and java.lang.**Error**

java.lang.**Exception** is superclass of java.lang.**RuntimeException, IOException, SQLException,** [**BrokenBarrierException**](http://www.javamadesoeasy.com/2015/03/cyclicbarrier-in-java.html)and many more other classes in java.

java.lang.**[RuntimeException](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)**is superclass of java.lang.[**NullPointerException**](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html)**, ArithmeticException** and many more other classesin java.

java.lang.**[Error](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html)**is superclass of java.lang.**VirtualMachineError, IOError, AssertionError,** [**ThreadDeath**](http://www.javamadesoeasy.com/2015/04/threaddeath-error-calling-stop-method.html)and many more other classesin java.

java.lang.**VirtualMachineError** is superclass of java.lang.**OutOfMemoryError, StackOverflowError** and many more other classesin java.

**Question 3. What are differences between checked and unchecked exceptions?**

**Answer**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***checked exception*** | ***unchecked exception*** |
| 1 | Also known as | **checked** exceptions are also known as **compileTime** exceptions. | **unchecked** exceptions are also known as **runtime** exceptions. |
| 2 | Should be solved at compile or runtime? | Checked exceptions are those which need to be taken care at compile time. | Unchecked exceptions are those which need to be taken care at runtime. |
| 3 | Benefit/ Advantage | We cannot proceed until we fix compilation issues which are most likely to happen in program, this helps us in avoiding runtime problems upto lot of extent. | Whenever runtime exception occurs execution of program is interrupted, but by handling these kind of exception we avoid such interruptions and end up giving some meaningful message to user. |
| 4 | Creating custom/own exception | |  | | --- | | **class** UserException **extends Exception** {     UserException(String s) {  **super**(s);     }  } |   By extending java.lang.**Exception**, we can create checked exception. | |  | | --- | | **class** UserException **extends RuntimeException** {     UserException(String s) {  **super**(s);     }  } |   By extending java.lang.**RuntimeException**, we can create unchecked exception. |
| 5 | [Exception propagation](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) | For **propagating checked** exceptions method must throw exception by using **throws** keyword. | **unchecked** exceptions are [**automatically propagated**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)in java. |
| 6 | handling checked and unchecked exception while overriding superclass method | *If superclass method throws/declare* ***checkedexception*** *>*   * overridden method of subclass **can** declare/**thrownarrower** (subclass of) **checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_93.html)), or * overridden method of subclass **cannot** declare/**throwbroader** (superclass of) **checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_94.html)), or * overridden method of subclass **can** declare/**throw any unchecked /RuntimeException** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_37.html)) | *If superclass method throws/declare* ***unchecked****>*   * overridden method of subclass **can** declare/**throw any unchecked /RuntimeException (superclass or subclass)** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of.html)), or * overridden method of subclass **cannot** declare/**throwany checked exception** (As shown in [**Program**](http://www.javamadesoeasy.com/2015/05/program-to-show-overridden-method-of_6.html)), |
|  | Which classes are which type of exception?  either  **checked orunchecked** exception? | The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions. | The class **RuntimeException and all its subclasses** are unchecked exceptions.  Likewise,  The class **Error and all its subclasses** are unchecked exceptions. |
| 7 | Most frequently faced exceptions | [SQLException](http://www.javamadesoeasy.com/2015/05/sqlexception-in-java.html),  IOException,  ClassNotFoundException | [NullPointerException](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html),  ArithmeticException ArrayIndexOutOfBoundsException. |

Read more on : [**Checked (compile time exceptions) and UnChecked (RuntimeExceptions) in java - Definition and differences**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)

**Question 4. What are 5 exception handling keywords in java?**

**Answer**. ***5*** [***keyword***](http://www.javamadesoeasy.com/2015/05/keywords-in-java-language.html) ***in java exception handling***

* + [**try**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) **- Any exception occurring in try block is catched by catch block.**

* + [**catch**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) **-** catch block is always followed by try block.

* + [**finally**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html)***finally block*** *can can only exist if try or try-catch block is there, finally block can’t be used alone in java.*

***Features*** *of finally >*

* finally block is **always executed** irrespective of exception is thrown or not.
  + finally is **keyword** in java.
  + finally block is optional in java, we may use it or not.

*finally block is* ***not executed*** *in following scenarios >*

* finally is not executed when **System.exit** is called.
* if in case **JVM crashes** because of some java.util.[**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html).

* + [**throw**](http://www.javamadesoeasy.com/2015/05/throw-exception-in-java.html) **throw** is a **keyword** in java.
    - **throw** keyword allows us to throw [**checked** or **unchecked**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)[exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html).

* + [**throws**](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html)**throws** is written in method’s definition to indicate that method can throw [exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html).

**Question 5. Explain what is Error in java?**

**Answer**. ***java.lang.[Error](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html)***

* Error is a subclass of **Throwable**
* Error **indicates some serious problems** that our **application should not try to catch.**
* Errors are **abnormal conditions in application**.
* Error and its subclasses are regarded as **unchecked** exceptions

Must know :

[ThreadDeath](http://www.javamadesoeasy.com/2015/04/threaddeath-error-calling-stop-method.html) is an error which application must not try to catch but it is normal condition.

**Question 6. What are differences between Exception and Error in java?**

**Answer**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | [**Exception**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) | [**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html) |
| 1 | serious problem? | Exception does **not indicate any serious problem**. | Error **indicates some serious problems** that our **application should not try to catch.** |
| 2 | divided into  [**checked** and **unchecked**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) | Exception are divided into **checked** and **unchecked exceptions**. | Error are **not divided** further into such classifications. |
| 3 | Which classes are which type of exception? either  **checked orunchecked** exception? | The class **Exception and all its subclasses** that are **not also subclasses of RuntimeException** are checked exceptions.  The class **RuntimeException and all its subclasses** are unchecked exceptions.  Likewise,  The class **Error and all its subclasses** are unchecked exceptions. | Error and its subclasses are regarded as **unchecked** exceptions |
| 4 | Most frequently faced exception and errors | **checked exceptions>**  SQLException,  IOException,  ClassNotFoundException  **unchecked exceptions>**  [NullPointerException](http://www.javamadesoeasy.com/2015/05/nullpointerexception-in-java.html), ArithmeticException, | **VirtualMachineError, IOError, AssertionError,** [**ThreadDeath**](http://www.javamadesoeasy.com/2015/04/threaddeath-error-calling-stop-method.html),  **OutOfMemoryError, StackOverflowError.** |
| 5 | Why to catch or not to catch? | Application **must catch** the Exception because they does not cause any major threat to application. | Application **must not catch** the Error because they does cause any major threat to application.  Example >  Let’s say errors like OutOfMemoryError and StackOverflowError occur and are caught then JVM might not be able to free up memory for rest of application to execute, so it will be better if application don’t catch these errors and is allowed to terminate. |

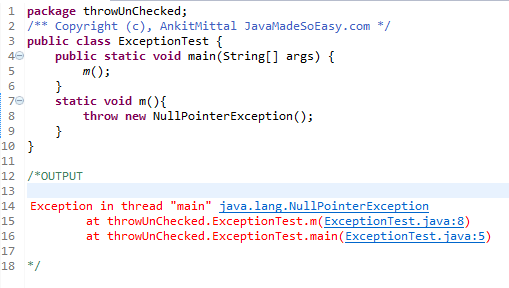
**Question 7. Explain throw keyword in java?**

**Answer**.

* [**throw**](http://www.javamadesoeasy.com/2015/05/throw-exception-in-java.html)is a **keyword** in java.
* **throw** keyword allows us to throw [**checked** or **unchecked**](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)[exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html).

***throw unchecked*** *exception >*

* We need **not to handle** unChecked exception either by catching it or throwing it.



We throw NullPointerException (unChecked exception) and didn’t handled it, no compilation error was thrown.

***throw checked*** *exception >*

* We need to handle checked exception either by catching it, or
* throwing it by using **throws** keyword. (When thrown, exception must be handled in calling environment)

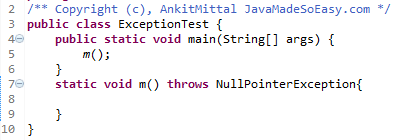
**Question 8. Explain throws keyword in java?**

**Answer**.

[throws](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html) is written in method’s definition to indicate that method can throw [exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html).

***throws*** [***unChecked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception >*

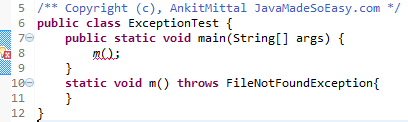
* We need **not to handle** unChecked exception either by catching it or throwing it.



Above code **throws** NullPointerException (unChecked exception) and didn’t handled it from where method m() was called and no compilation error was thrown.

***throws Checked*** *exception >*

* We need **to handle** checked exception either by catching it or throwing it further, if not handled we will face compilation error.



**Question 9. What is difference between throw and throws in java?**

**Answer**.

|  |  |  |
| --- | --- | --- |
|  | [**throw**](http://www.javamadesoeasy.com/2015/05/throw-exception-in-java.html) | [**throws**](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html) |
| 1 | **throw**[keyword](http://www.javamadesoeasy.com/2015/05/keywords-in-java-language.html) is used to throw an [exception](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) explicitly. | **throws**keyword is used to declare an exception. |
| 2 | **throw**is used **inside method**.  Example >  **staticvoid** m(){  **thrownew** FileNotFoundException();  } | **throws**is used **inmethod declaration**.  Example >  **staticvoid** m() throws FileNotFoundException{  } |
| 3 | **throw**is always **followed byinstanceof** Exception class.  Example >  **thrownew** FileNotFoundException() | **throws**is always **followed by name of Exception class**.  Example >  **throws**FileNotFoundException |
| 4 | **throw**can be used to throw **only one exception at time**.  Example >  **thrownew** FileNotFoundException() | **throws**can be used to throw **multiple exception at time**.  Example >  **throws**FileNotFoundException, NullPointerException  and many more... |
| 5 | **throw**cannot propagate exception to calling method.  https://lh4.googleusercontent.com/gO7Xp7MlqokGsbfiqCTCq0Px1YrtSGeLXjSP53nxwSHuKsTKmTmY2vd2g5Fd7shnyP3YRLNGrJcP7J5S1jfJPkdEWLOywBluAEoMQPKp3l4oIDHDK2ugUU4tvSmt_HDX5Q3DMBI | **throws**can [propagate exception](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) to calling method.  Please see these programs to understand how exception is propagated to calling method.  [**Program 1**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html) - Handling Exception by throwing it from m() method (using throws keyword) and handling it in try-catch block from where call to method m() was made.  [**Program 2**](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)- Throwing Exception from m() method and then again throwing it from calling method [ i.e. main method] |

**Question 10. How to create user defined checked and unchecked Exception in java?**

**Answer**.

*Creating user defined* [***checked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception >*

|  |
| --- |
| **class** UserDefinedException **extends Exception** {     UserDefinedException(String s) {  **super**(s);     }  } |

By extending java.lang.**Exception**, we can create checked exception.

*Creating user defined* [***unchecked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception >*

|  |
| --- |
| **class** UserDefinedException **extends RuntimeException** {     UserDefinedException(String s) {  **super**(s);     }  } |

By extending java.lang.**RuntimeException**, we can create unchecked exception.

**Question 11. How to use try-catch-finally ? Can we use try,catch or finally block alone?**

**Answer**.

*We can enclose exception prone code in >*

* [**try-catch** block](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html), or
* **try-finally** block, or
* **try-catch-finally** block.

*Using try-catch block*

|  |
| --- |
| **try{**  **//Code to be enclosed in try-catch block**  **}catch(Exception e){**  **}** |

Using try-finally block

|  |
| --- |
| **try{**  **//Code to be enclosed in try-finally block**  **}finally{**  **}** |

We cannot use **try block** alone, it must be followed by either **catch** or **finally**.

Using only try block will cause*compilation error*

|  |
| --- |
| **try{**  **//only try block will cause compilation error**  **}** |

*Likewise, we cannot use* ***catch block*** *alone, it always follows* ***try block.***

Using only catch block will cause*compilation error*

|  |
| --- |
| **catch{**  **//only catch block will cause compilation error**  **}** |

*Likewise, we cannot use* ***finally block*** *alone, it always follows* ***try block.***

Using only finally block will cause*compilation error*

|  |
| --- |
| **finally{**  **//only finally block will cause compilation error**  **}** |

**Question 12. Is it allowed to use multiple catch block in java?**

**Answer**. [Java exception handling](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) allows us to use [multiple catch block](http://www.javamadesoeasy.com/2015/05/multiple-catch-block-in-java.html).

**Important** Point  about **multiple catch block**>

1. **Exception class handled in starting catch block must be subclass of Exception class handled in following catch blocks (otherwise we will face compilation error).**
2. Either one of the multiple catch block will handle exception at time.

Program - Let’s understand the concept of multiple catch block>

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  publicclass ExceptionTest {     publicstaticvoid main(String[] args) {              try{                   inti=10/0; //will throw ArithmeticException            }**catch**(**ArithmeticException** ae){                   System.*out*.println("Exception handled - ArithmeticException");            }**catch**(**RuntimeException** re){                   System.*out*.println("Exception handled - RuntimeException");            }**catch**(**Exception** e){                   System.*out*.println("Exception handled - Exception");            }     }  }  /\*OUTPUT  Exception handled - ArithmeticException  \*/ |

In the above above >

**ArithmeticException** has been used in **first** catch block

**RuntimeException** has been used in **second** catch block

**Exception** has been used in **third** catch block

**Exception** is superclass of **RuntimeException** and

**RuntimeException** is superclass of **ArithmeticException.**

**Question 13. What is Automatic resource management in java 7?**

**Answer**.As we know java allows us to handle multiple exceptions by using [multiple catch blocks](http://www.javamadesoeasy.com/2015/05/multiple-catch-block-in-java.html). Now, java 7 has done improvements in multiple exception handling by introducing **multi catch syntax** which helps in [automatic resource management](http://www.javamadesoeasy.com/2015/05/catch-block-and-automatic-resource.html)**.**

*Features of* ***multi catch syntax*** *>*

* Has **improved way of catching multiple** [**exceptions**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html)**.**
* This syntax does **not looks clumsy**.
* **Reduces developer efforts** of writing multiple catch blocks.
* Allows us to **catch more than one exception in one catch block**.

Here is the **multi catch syntax**>

|  |
| --- |
| **try**{                   //code . . . . .            }**catch**(IOException **|** SQLException ex){                   //code . . . . .            } |

We could separate different exceptions using **pipe** ( **|** )

**Question 14. Explain try-with-resource in java?**

**Answer**. **Before java 7**, we used to write **explicit code for closing file in** [**finally**](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) **block by using** [**try-finally block**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html)like this >

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **publicclass** TryWithResourseTest {  **publicstaticvoid** main(String[] args) **throws** IOException {            InputStream inputStream = **null**;  **try**{                   inputStream = **new** FileInputStream("c:/txtFile.txt");                   //code......            }**finally**{  **if**(inputStream!=**null**)  **inputStream.close();**            }     }  } |

**In java 7**, using **Try-with-resources >**

* we need not to write **explicit code for closing file**.

|  |
| --- |
| **import** java.io.FileInputStream;  **import** java.io.IOException;  **import** java.io.InputStream;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **publicclass** TryWithResourseTest {  **publicstaticvoid** main(String[] args) **throws** IOException {  **try (InputStream inputStream = new FileInputStream("c:/txtFile.txt"))** {               //code...           }    }  } |

*Using multiple resources inside* ***Try-with-resources is also allowed.***

**Question  15. Now, question comes why we need not to close file when we are using Try-with-resources?**

**Answer**.  Because **FileInputStream** implements java.lang.**AutoCloseableinterface** (**AutoCloseable** interface’s close method automatically closes resources which are no longer needed.)

Which classes can be used inside **Try-with-resources?**

All the classes which implements **AutoCloseable** interface can be used inside **Try-with-resources.**

**Question 16. Explain finally keyword in java?**

**Answer**.

*try or* [*try-catch*](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) *block can be followed by finally block >*

* try**-finally** block, or

|  |
| --- |
| **try{**  **//Code to be enclosed in try-finally block**  **}finally{**  **}** |

* try-catch**-finally** block.

|  |
| --- |
| **try{**  **//Code to be enclosed in try-catch-finally block**  **}catch(Exception e){**  **}finally{**  **}** |

***finally block*** *can can only exist if try or try-catch block is there, finally block can’t be used alone in java.*

***Features*** *of finally >*

* finally block is **always executed** irrespective of exception is thrown or not.
* finally is [**keyword**](http://www.javamadesoeasy.com/2015/05/keywords-in-java-language.html)in java.

*finally block is* ***not executed*** *in following scenarios >*

* finally is not executed when **System.exit** is called.
* if in case **JVM crashes** because of some java.util.[**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html).

*Application of finally block in java programs >*

* We may use finally block to execute code for **database connection closing**, because closing connection in try or catch block may not be safe.
  + **Why closing connection in try block may not be safe?**
  + Because exception may be thrown in try block before reaching connection closing statement.

* + **Why closing connection in catch block may not be safe?**
  + Because inappropriate exception may be thrown in try block and we might not enter catch block to close connection.

For programs to demonstrate finally. [Please refer this post](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html).

**Question 17. Is it allowed to use nested try-catch in java?**

**Answer**. Java exception handling allows us to use [nested try-catch block](http://www.javamadesoeasy.com/2015/05/nested-try-catch-block-in-java.html).

Nested [try-catch](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) block means using try-catch block inside another try-catch block.

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **publicclass** ExceptionTest {  **publicstaticvoid** main(String[] args) {    **try**{  **int**i=10/0; //will throw ArithmeticException            }**catch**(ArithmeticException ae){                   System.*out*.println("try-catch block handled - ArithmeticException");    **//using nested try-catch block**  **try{**  **String s=null;**  **s.charAt(0); //will throw NullPointerException**  **}catch(NullPointerException npe){**  **System.*out*.println("NESTED try-catch block handled - "**  **+ "NullPointerException");**  **}**              }     }  } |

**Question 18. Discuss which checked and unchecked exception can be thrown/declared by subclass method while overriding superclass method in java?**

**Answer**.

*If superclass method throws/declare* ***unchecked/RuntimeException****>*

* overridden method of subclass **can** declare/**throw any unchecked /RuntimeException (superclass or subclass)**, or
* overridden method of subclass **cannot** declare/**throwany checked exception**, or
* overridden method of subclass **can** declare/**throwsame exception**, or
* overridden method of subclass **may not** declare/**throw any exception**.

*If superclass method throws/declare* ***checked****/****compileTimeexception*** *>*

* overridden method of subclass **can** declare/**thrownarrower** (subclass of) **checked exception**, or
* overridden method of subclass **cannot** declare/**throwbroader** (superclass of) **checked exception**, or
* overridden method of subclass **can** declare/**throw any unchecked /RuntimeException**, or
* overridden method of subclass **can** declare/**throwsame exception**, or
* overridden method of subclass **may not** declare/**throw any exception**.

*If superclass method does* ***not throw****/declare any* ***exception*** *>*

* overridden method of subclass **can** declare/**throw any unchecked /RuntimeException** , or
* overridden method of subclass **cannot** declare/**throwany checked exception**, or
* overridden method of subclass **may not** declare/**throw any exception**.

For programs please refer >[**Throw/declare checked and unchecked exception while overriding superclass method in java**](http://www.javamadesoeasy.com/2015/05/throwdeclare-checked-and-unchecked.html)

**Question 19. What will happen when catch and finally block both return value, also when try and finally both return value?**

**Answer**.

When **catch and finally block** both return value, **method will ultimately return value returned by** [**finally**](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) block irrespective of value returned by [catch](http://www.javamadesoeasy.com/2015/05/catch-block-and-automatic-resource.html) block.

|  |
| --- |
| /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **publicclass** ExceptionTest {  **publicstaticvoid** main(String[] args) {            System.*out*.println("method return -> "+*m*());     }    **static** String m(){  **try**{  **int** i=10/0; //will throw ArithmeticException            }**catch**(ArithmeticException e){  **return**"catch";            }**finally**{  **return**"finally";            }       }  }  /\*OUTPUT  method return -> finally  \*/ |

In above program, i=10/0 will throw ArithmeticException and enter catch block to return "catch", but ultimately control will enter finally block to return "finally".

Likewise, when [**try and finally**](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) **block** both return value, **method will ultimately return value returned by finally block** irrespective of value returned by try block. For program [please refer](http://www.javamadesoeasy.com/2015/05/what-will-happen-when-catch-and-finally.html).

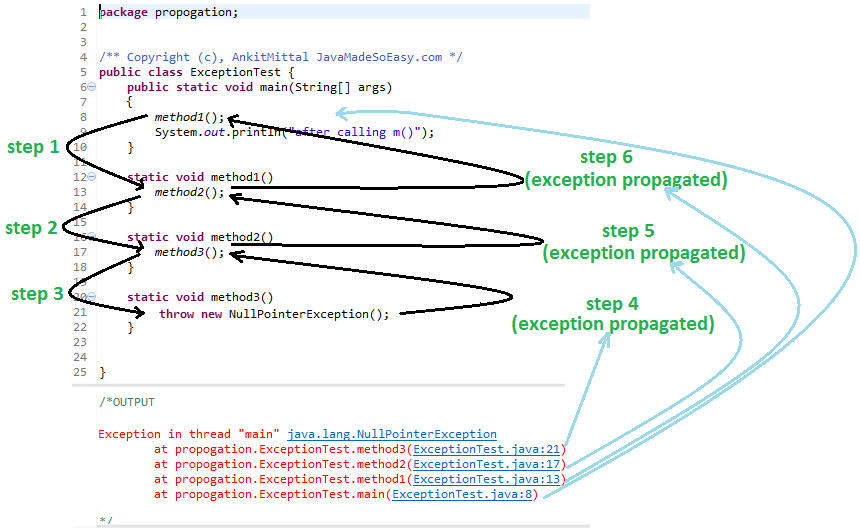
**Question 20. What is exception propagation in java?**

**Answer**. Whenever methods are called [stack](http://javamadesoeasy.com/2015/01/stacks.html) is formed and an exception is first thrown from the top of the stack and if it is not caught, it starts coming down the stack to previous methods until it is not caught.

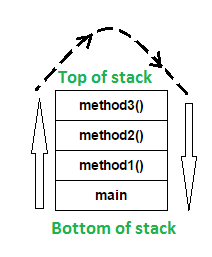
If exception remains uncaught even after reaching bottom of the stack it is propagated to JVM and program is terminated.

*Propagating* [***unchecked***](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html)*exception (NullPointerException) >*

**unchecked** exceptions are **automatically propagated** in java.

[](http://www.javamadesoeasy.com/2015/05/exception-propagation-in-java-deep.html)

[***stack***](http://javamadesoeasy.com/2015/01/stacks.html)*of methods is formed >*

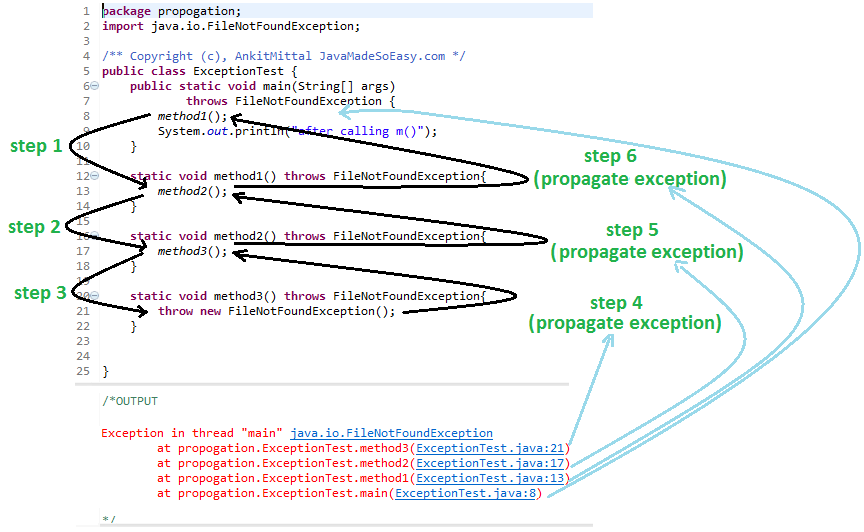


In the above program, stack is formed and an exception is first thrown from the top of the stack [ **method3()** ] and it remains uncaught there, and starts coming down the stack to previous methods to **method2()**,then to **method1()**,than to **main()** and it remains uncaught throughout.

exception remains uncaught even after reaching bottom of the stack [ **main()** ] so it is propagated to JVM and ultimately program is terminated by throwing exception [ as shown in output ].

*Propagating* ***checked*** *exception (FileNotFoundException) using throws keyword >*

For **propagating checked** exceptions method must throw exception by using [**throws**](http://www.javamadesoeasy.com/2015/05/throws-exception-in-java.html)keyword.



**Question 21. Can a catch or finally block throw exception?**

**Answer**. Yes, catch or finally block can throw checked or unchecked exception but it must be handled accordingly. Please refer this post for [handling checked and unchecked exceptions](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html).

**Question 22. Why shouldn’t you use Exception for catching all exceptions in java?**

**Answer**. Catching Exception rather than handling specific exception can be vulnerable to our application. [Multiple catch blocks](http://www.javamadesoeasy.com/2015/05/multiple-catch-block-in-java.html) must be used to catch specific exceptions, because handling specific exception gives developer the liberty of taking appropriate action and develop robust application.

**Question 23. What is Difference between** [**multiple catch block**](http://www.javamadesoeasy.com/2015/05/multiple-catch-block-in-java.html) **and** [**multi catch syntax**](http://www.javamadesoeasy.com/2015/05/catch-block-and-automatic-resource.html)**?**

**Answer**.

|  |  |  |
| --- | --- | --- |
|  | **multiple catch block** | **multi catch syntax** |
| 1 | multiple catch blocks were introduced in prior versions of Java 7 and does not provide any automatic resource management. | **multi catch syntax was introduced in** java 7 for improvements in multiple exception handling which helps in **automatic resource management.** |
| 2 | Here is the syntax for writing **multiple catch block**>   |  | | --- | | **try**{  //code . . . . .  }**catch**(**IOException** ex1){  //code . . . . .  } **catch**(**SQLException** ex2){  //code . . . . .  } | | Here is the **multi catch syntax**>   |  | | --- | | **try**{  //code . . . . .  }**catch**(IOException **|** SQLException ex){  //code . . . . .  } |   We could separate different exceptions using **pipe** ( **|** ) |
| 3 | For catching IOException and SQLException we need to write **two catch block** like this >  https://lh4.googleusercontent.com/Y8Pt2V80aY5BXlcs1viypo_8NBeNoVmA1Awad1o9oGIQhY02xSTW3M2fGrPGK0THNJP6yBczP-QuNHAiJjR9fMSGm1uawbwgaslhJh4KOihK-gbhHaBsPD7UYSsU2inbOxb4JPdn | with the help of multi catch syntax we can catch IOException and SQLException in one catch block using **multi catch syntax** like this >  https://lh5.googleusercontent.com/EDfjTrfY38x6H8uhRZ4ebbMwEdssvXcHaVb7dRVIr7r58vkNW4rtA8dpntIaHul_whEsS5no1B2EaawfQPTwwmOIyhjX-f0gAaRPzIHCpY36LafMmBjFpb6qPM4rQEhVb-XCEk0W |
| 4 | **When multiple catch blocks** are used , first catch block could be subclass of Exception class handled in following catch blocks like this >  IOException is subclass of Exception.https://lh5.googleusercontent.com/dDTyOichrPKc2g2d_KU_BLjUdPs1LpRqNum51I_x0iSD4KRaXe4T-gHeXLHEUELu4vE3W1jYt2ifV80dl0ZMaZZeRBYuIqSekdNpbUW_LdoQ30ms9gSi5Oj8kY43yAzcly5m29UC | If **Multi catch syntax** is used to catch subclass and its superclass than compilation error will be thrown.  IOException and Exception in **multi catch syntax** will cause compilation error “The exception **IOException** is already caught by the alternative **Exception**”.  https://lh4.googleusercontent.com/SFnwBVq0EsP5hKSaU7EdH35tTemyCbiqh35H-A_yd2KPEvoyzVJ7WU0y3yyafvAa0lFqSFW40Gm4b64_YaiopsmSx-hObaF6EoTNEhKihbyCIFwCb0k0lTyLU3F1pPjd5YNxuimg  **Solution >**  We must use only **Exception** to catch its subclass like this >  https://lh3.googleusercontent.com/ClWKvKo_sJKavB7eyyAgwQOHmkx_uCo3xZGxhX3_9kRUaavmPVd6dN6MM3Ix7HJ7m9ZXd5pbwyLwZ3ex8raeVavEkunmc3OyWemcUmra2fuaNtSZG1pNqz58wmSepXcuaUnaU2QF |
| 5 | Does not provide such features. | *Features of* ***multi catch syntax*** *>*   * Has **improved way of catching multiple** [**exceptions**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html)**.** * This syntax does **not looks clumsy**. * **Reduces developer efforts** of writing multiple catch blocks. * Allows us to **catch more than one exception in one catch block**. * Helps in **automatic resource management.** |

*For more read :*[***Difference between multiple catch block and multi catch syntax***](http://www.javamadesoeasy.com/2015/05/difference-between-multiple-catch-block.html)

**Question 24.  can a method be overloaded on basis of  exceptions ?**

**Answer**. Yes a method be overloaded on basis of  exceptions.

But now question which overloaded exception will be called.

Let’s take an example :

***Ques****. Let's say one method handles Exception and other handles ArithmeticException. Which method will be invoked when ArithmeticException is thrown?*

**Ans**. Method which handles more specific exception will be called.

Program >

|  |
| --- |
| **import** java.io.IOException;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com  \* Main class \*/  **publicclass** ExceptionTest {    **void** method(Exception e){            System.*out*.println(e+" caught in Exception method");     }  **void** method(ArithmeticException ae){            System.*out*.println(ae+" caught in ArithmeticException method");     }    **publicstaticvoid** main(String[] args) {            ExceptionTest obj=**new** ExceptionTest();            obj.method(**new** ArithmeticException());            obj.method(**new** IOException());     }  }  /\* OUTPUT  java.lang.ArithmeticException caught in ArithmeticException method  java.io.IOException caught in Exception method  \*/ |

**Question 25.  Mention few exception handling best practices ?**

**Answer**.

* [**Throw**](http://www.javamadesoeasy.com/2015/05/throw-exception-in-java.html)[**exceptions**](http://www.javamadesoeasy.com/2015/05/exception-handling-exception-hierarchy.html) **when the method cannot handle the exception**, and more **importantly**, must be **handled by the caller**.

**Example -**

In Servlets - doGet() and doPost() throw ServletException or IOException in certain circumstances where the request could not be read correctly. Neither of these methods are in a position to handle the exception, but the container is (which generally results in the 404 error page in most cases).

* **Bubble the exception if the method cannot handle it**. This is a corollary of the above point, but applicable to methods that must catch the exception. If the caught exception cannot be handled correctly by the method, then it is preferable to bubble it.

* **Throw the exception right away**. If an exception scenario is encountered, then it is a good practice to throw an exception indicating the original point of failure, instead of attempting to handle the failure via error codes, until a point deemed suitable for throwing the exception. In other words, attempt to minimize mixing exception handling with error handling.

* **Either log the exception or bubble it, but don't do both**. Logging an exception often indicates that the exception stack has been completely unwound, indicating that no further bubbling of the exception has occurred. Hence, it is not recommended to do both at the same time, as it often leads to a frustrating experience in debugging.
* **Application should not try to catch** [**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html) **-** Because, in most of cases recovery from an Error is almost impossible. So, application must be allowed to terminate.

**Example>**

Let’s say errors like OutOfMemoryError and StackOverflowError occur and are caught then JVM might not be able to free up memory for rest of application to execute, so it will be better if application don’t catch these errors and is allowed to terminate.

Read : Complete list of [**Exception handling best practices and guidelines for using exceptions in java**](http://www.javamadesoeasy.com/2015/05/exception-handling-best-practices-and.html)

**Question 26.  Difference between Final, Finally and Finalize?**

**Answer**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | [***final***](http://www.javamadesoeasy.com/2015/05/final-keyword-in-java-20-salient.html) | [***finally***](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) | [***finalize***](http://www.javamadesoeasy.com/2015/05/finalize-method-in-java-10-salient.html) |
| **1** | [final](http://www.javamadesoeasy.com/2015/05/final-keyword-in-java-20-salient.html) can be applied to **variable**, **method** and **class** in java. | [finally](http://www.javamadesoeasy.com/2015/05/finally-block-in-java.html) is a block. | [finalize](http://www.javamadesoeasy.com/2015/05/finalize-method-in-java-10-salient.html) is a method. |
| **2** | ***2.1) Final variable***  **final memberVariable**  **final local variable**  **final static variable**  **Final memberVariable** of class must be initialized at time of declaration, once initialized final memberVariable cannot be assigned a new value.  Final variables are called **constants** in java.   |  | | --- | | **class** FinalTest {  **finalint**x=1; //memberVariable/instanceVariable  } |   If constructor is defined then final memberVariable can be initialized in constructor but  once initialized cannot be assigned a new value.   |  | | --- | | **class** FinalTest {  **finalint**x; //memberVariable/instanceVariable     FinalTest() {            x = 1; //final memberVariable can be initialized in constructor.     }  } |   **Final local variable** can be left uninitialized at time of declaration and can be initialized later, but once initialized cannot be assigned a new value.   |  | | --- | | **class** FinalTest {  **void** method(){  **finalint** x; //uninitialized at time of declaration        x=1;    }  } |   **Final static variable** of class must be initialized at time of declaration or can be initialized in static block, once initialized final static variable cannot be assigned a new value.  If static block is defined then final static variable can be initialized in static block, once initialized final static variable cannot be assigned a new value.   |  | | --- | | **class** FinalTest {  **finalstaticint***x*; //static variable  **static**{ //static block  *x*=1;   }  } |   ***2.2) Final method***  **Final method** cannot be overridden, any attempt to do so will cause compilation error.  https://lh6.googleusercontent.com/9O1O5MRwg1NVPsVCuL-i3-M2HsZ4vFfyEkajayflbtUXS3PMil96r-z69VJigDDJKXyy4KE8UJbosx66qyqbojmEcvGAW__dHg6oKVB8-_cus4a6r_peqKFO-hmqx8CWgoWIOgh8  Runtime polymorphism is not applicable on final methods because they cannot be inherited.  ***2.3) Final class***  **Final class** cannot be extended, any attempt to do so will cause compilation error.  https://lh3.googleusercontent.com/OWVRfY8Eqw9fRdxNxyFG6OAnxBnLo79ffiMTMK6vWzNgcb0yjyW9d5FzlIaRSNH9i4TazrkZxRnlN3spCOmw54iXR6AHwegwwF-3R1tZKHB0I3K88aCVaqHxvsWlr9ZFL5TYDbaK | *try or* [*try-catch*](http://www.javamadesoeasy.com/2015/05/try-catch-finally-block-in-java.html) *block can be followed by finally block >*  try**-finally** block, or   |  | | --- | | **try{**  **//Code to be enclosed in try-finally block**  **}finally{**  **}** |   try-catch**-finally** block.   |  | | --- | | **try{**  **//Code to be enclosed in try-catch-finally block**  **}catch(Exception e){**  **}finally{**  **}** |   ***finally block*** *can can only exist if try or try-catch block is there, finally block can’t be used alone in java.*  ***finally block is not executed in following scenarios >***  finally is not executed when **System.exit** is called.  if in case **JVM crashes** because of some java.util.[**Error**](http://www.javamadesoeasy.com/2015/05/javalangerror-in-exception-handling-in.html). | finalize method is called before garbage collection by JVM,  finalize method is called for any cleanup action that may be required before garbage collection.   |  | | --- | | /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  @Override  **protectedvoid finalize() throws Throwable** {  **try** {        System.*out*.println("in   finalize() method, "                                +   "doing cleanup activity");    } **catch** (Throwable throwable) {  **throw** throwable;  }  } |   finalize() method is defined **in java.lang.Object** |
| **3** | - | finally block can only exist if try or try-catch block is there, finally block can’t be used alone in java. | We can *force early garbage collection in java* by using following methods >  **System.*gc*(); Runtime.*getRuntime*().gc();**  **System.*runFinalization*(); Runtime.*getRuntime*().runFinalization();** |
| **4** | - | finally is always executed irrespective of exception thrown. | If any uncaught [exception](http://www.javamadesoeasy.com/2015/05/checked-compile-time-exceptions-and.html) is thrown inside finalize method -  **exception is ignored,**  **thread is terminated and**  **object is discarded.**  **Note :** Any exception thrown by the finalize method causes the finalization of this object to be halted, but is otherwise ignored. |
| **5** | - | Currently executing thread calls finally method. | JVM does not guarantee which [***daemon***](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html)[thread](http://www.javamadesoeasy.com/2015/03/what-is-thread-in-java.html) will invoke the finalize method for an object. |
| **6** | final is a keyword in java. | finally Is a keyword in java. | finalize is not a keyword in java. |

**Question 1. What is Thread in java?**

**Answer.**

* Threads **consumes CPU in best possible manner**, hence enables multi processing. Multi threading **reduces idle time of CPU** which improves performance of application.
* Thread are **light weight process**.

* A thread class belongs to **java.lang package**.
* We can create multiple threads in java, **even if we don’t create any Thread, one Thread at least  do exist** i.e. **main thread**.
* **Multiple threads run parallely in java.**
* Threads have their **own stack**.
* **Advantage** of Thread : Suppose one thread needs 10 minutes to get certain task, 10 threads used at a time could complete that task in 1 minute, because threads can run parallely.

*Also Read :* [***Top and most important Interview Questions and answers in Java***](http://www.javamadesoeasy.com/p/interview-questions.html)

**Question 2. What is difference between Process and Thread in java?**

**Answer.**  One process can have multiple Threads,

Thread are **subdivision** of Process. One or more Threads runs in the context of process. Threads can execute any part of process. And same part of process can be executed by multiple Threads.

Processes have their own **copy of the data segment of the parent process** while Threads have **direct access to the data segment of its process**.

Processes have their **own address** while Threads share the **address space of the process that created it**.

Process creation needs whole lot of stuff to be done, we **might need to copy whole parent process**, but Thread can be **easily created**.

Processes can **easily communicate with child processes** but **interprocess communication is difficult**. While, Threads **can easily communicate with other threads of the same process using** [**wait() and notify() methods**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html).

In process **all threads share system resource** like heap Memory etc. while Thread has its **own stack.**

Any change made to process **does not affect child processes**, but any change made to thread **can affect the behavior of the other threads of the process**.

[**Example** to see where threads on are created on different processes and same process.](http://www.javamadesoeasy.com/2015/03/when-threads-are-not-lightweight.html)

**Question 3. How to implement Threads in java?**

**Answer.**  This is very basic threading question. Threads can be created in two ways i.e. by [implementing **java.lang.Runnable** interface or extending **java.lang.Thread** class](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) and then extending run method.

Thread has its own variables and methods, it lives and dies on the heap. [But a thread of execution is an individual process that has its own call stack](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html). Thread are lightweight process in java.

1. Thread creation by  implementing**java.lang.Runnable**interface.

We will create object of class which implements Runnable interface :

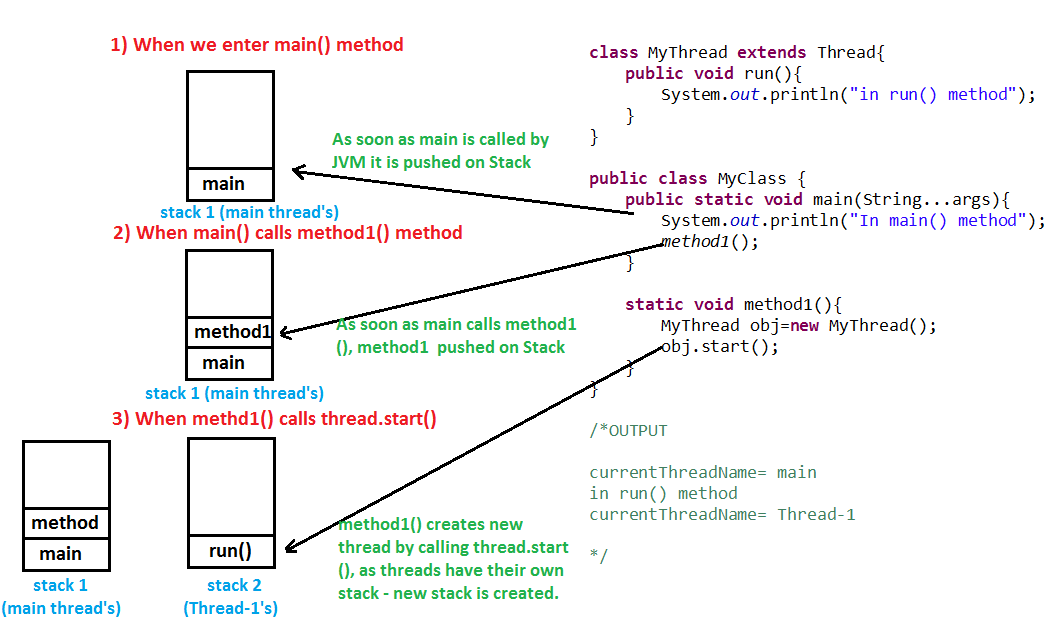
|  |
| --- |
| **MyRunnable runnable=new MyRunnable();**  **Thread thread=new Thread(runnable);** |

     2) And then create Thread object by calling constructor and passing reference of Runnable interface i.e.  **runnable** object :

|  |
| --- |
| **Thread thread=new Thread(runnable);** |

**Question 4 . Does Thread implements their own Stack, if yes how? (Important)**

**Answer.**  **Yes**, [Threads have their own stack](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html). This is very interesting question, where interviewer tends to check your basic knowledge about how [threads internally maintains their own stacks](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html). I’ll be explaining you the concept by diagram.

[](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html)

**Question 5. We should implement Runnable interface or extend Thread class. What are differences between implementing Runnable and extending Thread?**

**Answer.** Well the answer is you must [**extend Thread**](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) only when you are looking to **modify run() and other methods as well**. If you are simply looking to **modify only the run() method** [**implementing Runnable**](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) **is the best option (**Runnable interface has only one abstract method i.e. run() **)**.

[*Differences between implementing Runnable interface and extending Thread class*](http://www.javamadesoeasy.com/2015/03/differences-between-implementing.html) *-*

1. ***Multiple inheritance in not allowed in java :*** When we [implement Runnable](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) interface **we can extend another class as well**, but if we extend Thread class **we cannot extend any other class** because java does not allow multiple inheritance. So, same work is done by implementing Runnable and [extending Thread](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) but in case of implementing Runnable we are still left with option of extending some other class. **So, it’s better to implement Runnable.**
2. [***Thread safety***](http://www.javamadesoeasy.com/2015/03/guidelines-to-thread-safe-code-most.html) ***:*** When we implement Runnable interface, **same object is shared amongst multiple threads**, but when we extend Thread class **each and every thread gets associated with new object**.
3. ***Inheritance (Implementing Runnable is lightweight operation) :*** When we extend Thread **unnecessary all Thread class features are inherited**, but when we implement Runnable interface no **extra feature are inherited**, as Runnable only consists only of one abstract method i.e. run() method.  **So, implementing Runnable is lightweight operation.**
4. ***Coding to interface :*** Even **java recommends coding to interface**. So, we must implement Runnable rather than extending thread. Also, Thread class implements Runnable interface.
5. ***Don’t extend unless you wanna modify fundamental behaviour of class, Runnable interface has only one abstract method i.e. run()  :*** We must [**extend Thread**](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) only when you are looking to **modify run() and other methods as well**. If you are simply looking to **modify only the run() method** [**implementing Runnable**](http://www.javamadesoeasy.com/2015/03/implementing-threads-in-java-by.html) **is the best option (**Runnable interface has only one abstract method i.e. run() **). We must not extend Thread class unless we're looking to modify fundamental behaviour of Thread class.**
6. ***Flexibility in code when we implement Runnable :*** When we extend Thread first a fall all thread features are inherited and **our class becomes direct subclass of Thread , so whatever action we are doing is in Thread class**. But, when we implement Runnable **we create a new thread and pass runnable object as parameter,we could pass runnable object to executorService & much more**. So, we have more options when we implement Runnable and **our code becomes more flexible.**
7. ***ExecutorService :*** If we implement Runnable, **we can start multiple thread created on runnable object  with ExecutorService** (because we can start Runnable object with new threads), **but not in the case when we extend** Thread (because thread can be started only once).

**Question 6. How can you say Thread behaviour is unpredictable? (Important)**

**Answer.** The solution to question is quite simple, [Thread behaviour is unpredictable](http://www.javamadesoeasy.com/2015/03/thread-behaviour-is-unpredictable.html) because execution of Threads depends on Thread scheduler, thread scheduler may have different implementation on different platforms like windows, unix etc. Same threading program may produce different output in subsequent executions even on same platform.

To achieve we are going to create 2 threads on same Runnable Object, create for loop in run() method and start  both threads. There is no surety that which threads will complete first,  both threads will enter anonymously in for loop.

**Question 7 . When threads are not lightweight process in java?**

**Answer.** Threads are [**lightweight process**](http://www.javamadesoeasy.com/2015/03/when-threads-are-not-lightweight.html) **only if threads of same process are executing concurrently**. But **if threads of different processes are executing concurrently then threads are** [**heavy weight process**](http://www.javamadesoeasy.com/2015/03/when-threads-are-not-lightweight.html).

**Question 8. How can you ensure all threads that started from main must end in order in which they started and also main should end in last? (Important)**

**Answer.**  Interviewers tend to know interviewees knowledge about Thread methods. So this is time to prove your point by answering correctly. We can use [**join() method**](http://www.javamadesoeasy.com/2015/03/join-method-ensure-all-threads-that.html)to ensure all threads that started from main must end in order in which they started and also main should end in last.In other words **waits for this thread to die**. **Calling join() method internally calls join(0);**

[**DETAILED DESCRIPTION :** Join() method - ensure all threads that started from main must end in order in which they started and also main should end in last. Types of join() method with programs- 10 salient features of join.](http://www.javamadesoeasy.com/2015/03/join-method-ensure-all-threads-that.html)

**Question 9.What is difference between starting thread with run() and start() method? (Important)**

**Answer.** This is quite interesting question, it might confuse you a bit and at time may make you think is there really any [difference between starting thread with run() and start() method](http://www.javamadesoeasy.com/2015/03/difference-between-starting-thread-with.html).

When you **call start()** method, **main thread internally calls run() method** to start newly created Thread, so **run() method is ultimately called by newly created thread**.

When you **call run()** method **main thread** rather than starting run() method with newly thread it start **run() method by itself**.

**Question 10. What is significance of using** [**Volatile**](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html) **keyword? (Important)**

**Answer.** Java allows threads to **access shared variables**. As a rule, to ensure that **shared variables are consistently updated**, a thread should ensure that it has **exclusive use of such variables by obtaining a lock** that enforces mutual exclusion for those shared variables.

**If a field is declared** [**volatile**](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html)**, in that case the Java memory model ensures that all threads see a consistent value for the variable.**

Few small questions>

Q. Can we have [volatile](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html) methods in java?

1. **No**, volatile is only a keyword, can be used only with variables.

Q. Can we have synchronized variable in java?

1. **No**, synchronized can be used only with methods, i.e. in method declaration.

**DETAILED DESCRIPTION :** [Volatile keyword in java- difference between synchronized and volatile with programs, 10 key points about volatile keyword, why volatile variables are not cached in memory](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html)

**Question 11. Differences between synchronized and volatile keyword in Java? (Important)**

**Answer.**Its very important question from interview perspective.

1. [Volatile](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html) can be used as a **keyword** against the variable, we **cannot** use volatile against method declaration.

**volatilevoid** method1(){} //it’s illegal, compilation error.

**volatile** int i; **//legal**

While [synchronization](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) can be used in method declaration or we can create synchronization blocks (In both cases thread acquires lock on object’s monitor). Variables cannot be synchronized.

Synchronized method:

**synchronizedvoid** method2(){} //legal

Synchronized block:

**void** method2(){

**synchronized** (**this**) {

          //code inside synchronized block.

          }

}

Synchronized variable (illegal):

**synchronized** int i;//it’s illegal, compilatiomn error.

1. [Volatile](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html) does not acquire any lock on variable or object, but [Synchronization](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) acquires lock on method or block in which it is used.
2. [Volatile](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html) variables are not cached, but variables used inside [synchronized](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) method or block are cached.
3. When volatile is used will never create deadlock in program, as volatile never obtains any kind of lock . But in case if synchronization is not done properly, we might end up creating dedlock in program.
4. Synchronization may cost us performance issues, as one thread might be waiting for another thread to release lock on object. But volatile is never expensive in terms of performance.

**DETAILED DESCRIPTION :** [**Differences between synchronized and volatile keyword in detail with programs.**](http://www.javamadesoeasy.com/2015/03/differences-between-synchronized-and.html)

**Question 12. Can you again start Thread?**

**Answer.No**, [we cannot start Thread again](http://www.javamadesoeasy.com/2015/03/can-we-start-thread-again.html), doing so will throw runtimeException java.lang.IllegalThreadStateException. The reason is once run() method is executed by Thread, it goes into [**dead state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html).

Let’s take an example-

Thinking of starting thread again and calling start() method on it (which internally is going to call run() method) for us is some what like asking dead man to wake up and run. As, after completing his life person goes to **dead state**.

**Question 13. What is race condition in multithreading and how can we solve it? (Important)**

**Answer.** This is very important question, this forms the core of multi threading, you should be able to explain about [race condition in detail](http://www.javamadesoeasy.com/2015/03/race-condition-in-multithreading-and.html).

When more than one thread try to access same resource without synchronization causes race condition.

**So we can** [**solve race condition**](http://www.javamadesoeasy.com/2015/03/race-condition-in-multithreading-and.html) **by using either** [**synchronized block or synchronized method**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)**.** When no two threads can access same resource at a time phenomenon is also called as **mutual exclusion**.

**Few sub questions>**

What if two threads try to **read** same resource without [synchronization](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)?

When two threads try to read on same resource without synchronization, **it’s never going to create any problem**.

What if two threads try to **write** to same resource without [synchronization](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)?

When two threads try to **write** to same resource without synchronization, **it’s going to create synchronization problems**.

**Question 14. How threads communicate between each other?**

**Answer.** This is very must know question for all the interviewees, you will most probably face this question in almost every time you go for interview.

**Threads can communicate** with each other by using [**wait(), notify() and notifyAll()**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) methods.

**Question 15. Why wait(), notify()  and notifyAll() are in Object class and not in Thread class? (Important)**

**Answer.**

1. **Every Object has a monitor**, acquiring that monitors allow thread to hold lock on object. But **Thread class does not have any monitors**.
2. wait(), notify() and notifyAll()are called on objects only >**When wait() method** is called on object by thread **it waits for another thread** on that object to **release object monitor by calling** [**notify() or notifyAll()**](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html) method on that object.

**When notify() method is called** on object by thread **it notifies all the threads**

which are **waiting for that object monitor** that object monitor is available now.

So, this shows that wait(), notify() and notifyAll() are called on objects only.

[Now, Straight forward question that comes to mind is how thread acquires object lock by](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)

[acquiring object monitor? Let’s try to understand this basic concept in detail?](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)

1. Wait(), notify() and notifyAll() method being in Object class allows all the **threads created on that object** to **communicate** with other.  [As multiple threads may exist on same object].
2. As **multiple threads exists on same object**. Only one thread can hold object monitor at a time. As a result thread can notify other threads of same object that lock is available now. But, thread having these methods does not make any sense because multiple threads exists on object its not other way around (i.e. multiple objects exists on thread).
3. Now let’s discuss one **hypothetical** scenario, **what will happen if Thread class contains wait(), notify() and notifyAll() methods**?

Having wait(), notify() and notifyAll() methods **means Thread class also must have their monitor**.

Every thread having their monitor will create few problems -

>**Thread communication problem.**

>**Synchronization on object won’t be possible**- Because object has monitor, one object can have multiple threads and thread hold lock on object by holding object monitor. But if each thread will have monitor, we won’t have any way of achieving synchronization.

>**Inconsistency in state of object** (because synchronization won't be possible).

**Question 16. Is it important to acquire object lock before calling wait(), notify() and notifyAll()?**

**Answer.Yes**, it’s mandatory to acquire object lock before calling these methods on object. As discussed above [**wait(), notify()  and notifyAll()**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) **methods are always called from** [**Synchronized block**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) **only**, and **as soon as thread enters synchronized block it acquires object lock** (by holding object monitor). If we call these methods without acquiring object lock i.e. from outside synchronize block then java.lang. IllegalMonitorStateException is thrown at runtime.

Wait() method needs to enclosed in try-catch block, because it throws compile time exception i.e. InterruptedException.

**Question 17. How can you solve consumer producer problem by using wait() and notify() method? (Important)**

**Answer.**  Here come the time to answer **very very important question from interview perspective**. Interviewers tends to check how sound you are in threads inter communication. Because for solving this problem we got to **use synchronization blocks, wait() and notify() method very cautiously**. **If you misplace synchronization block or any of the method** that **may cause your program to go horribly wrong**. So, before going into this question first i’ll recommend you to understand how to use synchronized blocks, wait() and notify() methods.

**Key points** we need to ensure before programming :

**>**Producer will produce total of 10 products and cannot produce more than 2 products at a time until products are being consumed by consumer.

**Example**> when sharedQueue’s size is 2, wait for consumer to consume (consumer will consume by calling remove(0) method on sharedQueue and reduce sharedQueue’s size). As soon as size is less than 2, producer will start producing.

**>**Consumer can consume only when there are some products to consume.

**Example**> when sharedQueue’s size is 0, wait for producer to produce (producer will produce by calling add() method on sharedQueue and increase sharedQueue’s size).   As soon as size is greater than 0, consumer will start consuming.

Explanation of **Logic** >

We will create sharedQueue that will be shared amongst Producer and Consumer. We will now start consumer and producer thread.

Note: it does not matter order in which threads are started (because rest of code has taken care of synchronization and key points mentioned above)

First we will start consumerThread >

|  |
| --- |
| consumerThread.start(); |

consumerThread will enter run method and call consume() method. There it will check for sharedQueue’s size.

-if size is equal to 0 that means producer hasn’t produced any product, wait for producer to produce by using below piece of code-

|  |
| --- |
| **synchronized** (sharedQueue) {  **while** (sharedQueue.size() == 0) {                 sharedQueue.wait();          }        } |

-if size is greater than 0, consumer will start consuming by using below piece of code.

|  |
| --- |
| **synchronized** (sharedQueue) {            Thread.*sleep*((**long**)(Math.*random*() \* 2000));          System.*out*.println("consumed : "+ sharedQueue.remove(0));          sharedQueue.notify();         } |

Than we will start producerThread >

|  |
| --- |
| producerThread.start(); |

producerThread will enter run method and call produce() method. There it will check for sharedQueue’s size.

-if size is equal to 2 (i.e. maximum number of products which sharedQueue can hold at a time), wait for consumer to consume by using below piece of code-

|  |
| --- |
| **synchronized** (sharedQueue) {  **while** (sharedQueue.size() == maxSize) { //maxsize is 2        sharedQueue.wait();       }   } |

-if size is less than 2, producer will start producing by using below piece of code.

|  |
| --- |
| **synchronized** (sharedQueue) {        System.*out*.println("Produced : " + i);        sharedQueue.add(i);        Thread.*sleep*((**long**)(Math.*random*() \* 1000));        sharedQueue.notify();   } |

**DETAILED DESCRIPTION** [**with program : Solve Consumer Producer problem by using wait() and notify() methods in multithreading.**](http://www.javamadesoeasy.com/2015/03/solve-consumer-producer-pattern-by.html)

Another illustration with program : [How to solve **Consumer Producer** problem by using **wait() and notify()** methods, where **consumer can consume only when production is over**.](http://www.javamadesoeasy.com/2015/03/solve-consumer-producer-problem-by_2.html)

**Question 18.** [**How to solve Consumer Producer problem without using wait() and notify() methods, where consumer can consume only when production is over.**](http://www.javamadesoeasy.com/2015/03/how-to-solve-consumer-producer-problem.html)**?**

**Answer.** In this problem, producer will allow consumer to consume only when 10 products have been produced (i.e. when production is over).

We will approach by keeping one boolean variable **productionInProcess** and initially setting it to **true**, and later when production will be over we will set it to **false**.

**DETAILED DESCRIPTION** : [How to solve **Consumer Producer** problem **without using wait() and notify()** methods, where **consumer can consume only when production is over**.](http://www.javamadesoeasy.com/2015/03/how-to-solve-consumer-producer-problem.html)

**Question 19. How can you solve consumer producer pattern by using BlockingQueue? (Important)**

**Answer.** Now it’s time to gear up to face question which is most probably going to be followed up by previous question i.e. after how to solve consumer producer problem using wait() and notify() method. Generally you might wonder why interviewer's are so much interested in asking about [solving consumer producer problem using BlockingQueue](http://www.javamadesoeasy.com/2015/03/solve-consumer-producer-problem-by.html), answer is they want to know how strong knowledge you have about java concurrent Api’s, this Api use consumer producer pattern in very optimized manner, BlockingQueue is designed is such a manner that it offer us the best performance.

[**BlockingQueue is a interface** and we will use its **implementation class LinkedBlockingQueue**.](http://www.javamadesoeasy.com/2015/03/solve-consumer-producer-problem-by.html)

Key methods for solving consumer producer pattern are >

|  |
| --- |
| **put(i);**   //used by producer to put/produce in sharedQueue.  **take();** //used by consumer to take/consume from sharedQueue. |

**Question 20. What is** [**deadlock**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) **in multithreading? Write a program to form** [**DeadLock**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) **in multi threading and also how to solve DeadLock situation. What measures you should take to avoid deadlock? (Important)**

**Answer.**  This is very important question from interview perspective. But, what makes this question important is it checks interviewees capability of [**creating and detecting deadlock**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html). If you can write a code to form deadlock, than I am sure you must be well capable in solving that deadlock as well. If not, later on this post we will learn how to solve deadlock as well.

First question comes to mind is, [**what is deadlock in multi threading program**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html)**?**

**Deadlock is a situation where two threads are waiting for each other to release lock holded by them on resources.**

But how [deadlock](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) could be formed :

**Thread-1 acquires lock on String.class** and then calls [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) method which gives Thread-2 the chance to execute immediately after Thread-1 has acquired lock on String.class and **Thread-2 acquires lock on Object.class** then calls sleep() method and **now it waits for Thread-1 to release lock on String.class**.

**Conclusion:**

Now, **Thread-1 is waiting for Thread-2 to release lock on Object.class** and **Thread-2 is waiting for Thread-1 to release lock on String.class** and deadlock is formed.

|  |
| --- |
| **Code called by Thread-1**    **publicvoid** run() {  **synchronized** (String.**class**) {                     Thread.sleep(100);  **synchronized** (Object.**class**) {                   }            }  }  **Code called by Thread-2**    **publicvoid** run() {  **synchronized** (Object.**class**) {                     Thread.sleep(100);  **synchronized** (String.**class**) {                   }            }  } |

Here comes the **important** part, how above formed **deadlock** could be **solved** :

**Thread-1 acquires lock on String.class** and then calls [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) method which gives Thread-2 the chance to execute immediately after Thread-1 has acquired lock on String.class and **Thread-2 tries to acquire lock on String.class** but lock is holded by Thread-1. Meanwhile, Thread-1 completes successfully. As Thread-1 has completed successfully it releases lock on String.class, Thread-2 can now acquire lock on String.class and complete successfully without any deadlock formation.

**Conclusion:** No deadlock is formed.

|  |
| --- |
| **Code called by Thread-1**    **publicvoid** run() {  **synchronized** (String.**class**) {                     Thread.sleep(100);  **synchronized** (Object.**class**) {                   }            }  }  **Code called by Thread-2**    **publicvoid** run() {  **synchronized** (String.**class**) {                     Thread.sleep(100);  **synchronized** (Object.**class**) {                   }            }  } |

Few important measures to avoid [Deadlock](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html)>

1. **Lock specific member variables of class rather than locking whole class**: We must try to lock specific member variables of class rather than locking whole class.
2. **Use join() method:** If possible try touse join() method, although it may refrain us from taking full advantage of multithreading environment because threads will start and end sequentially, but it can be handy in avoiding deadlocks.
3. **If possible try avoid using nested synchronization blocks.**

**Question 21. Have you ever generated thread dumps or analyzed Thread Dumps? (Important)**

**Answer.** Answering this questions will show your in depth knowledge of Threads. Every experienced must know how to generate Thread Dumps.

[**VisualVM**](http://www.javamadesoeasy.com/2015/03/visualvm-thread-dumps-generating-and_74.html)  is most popular way to generate Thread Dump and is most widely used by developers. It’s important to understand usage of VisualVM for in depth knowledge of VisualVM. I’ll recommend every developer must understand this topic to become master in multi threading.

It helps us in analyzing threads performance, [thread states](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html), CPU consumed by threads, garbage collection and much more.  For detailed information see [**Generating and analyzing Thread Dumps using VisualVM - step by step detail to setup VisualVM with screenshots**](http://www.javamadesoeasy.com/2015/03/visualvm-thread-dumps-generating-and_74.html)

[**jstack**](http://www.javamadesoeasy.com/2015/03/jstack-thread-dumps-generating-and.html) is very easy way to generate Thread dump and is widely used by developers. I’ll recommend every developer must understand this topic to become master in multi threading. For creating Thread dumps we **need not to download any jar or any extra software**. For detailed information see [**Generating and analyzing Thread Dumps using JSATCK - step by step detail to setup JSTACK with screenshots**](http://www.javamadesoeasy.com/2015/03/jstack-thread-dumps-generating-and.html).

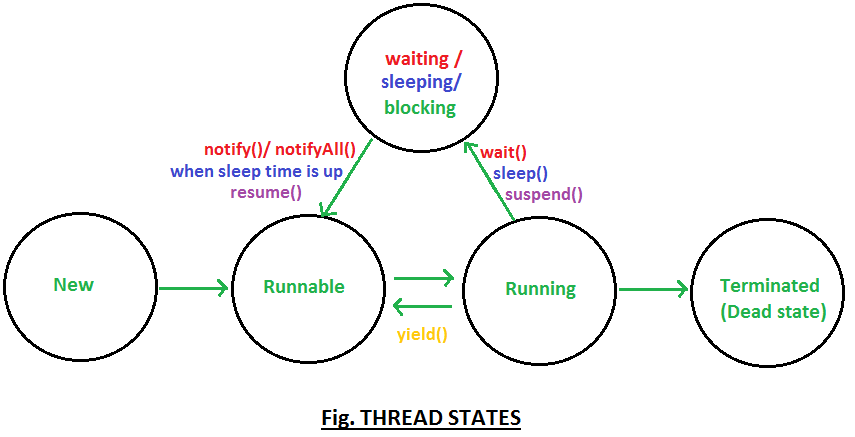
**Question 22. What is life cycle of Thread, explain thread states? (Important)**

**Answer.**  [**Thread states/ Thread life cycle**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **is very basic question, before going deep into concepts we must understand Thread life cycle.**

**Thread have following states >**

* **New**
* **Runnable**
* **Running**
* **Waiting**/**blocked/sleeping**
* **Terminated (Dead)**

**Thread states/ Thread life cycle in diagram >**

****

**Thread states in detail >**

**New : When instance of thread is created using new operator it is in new state**, but the start() method has not been invoked on the thread yet, thread is not eligible to run yet.

**Runnable :**  **When start() method is called on thread it enters runnable state**.

**Running :** Thread scheduler selects thread to go fromrunnable to running state. In running state Thread starts executing by entering run() method.

**Waiting/blocked/sleeping :** In this state a thread is not eligible to run.

>Thread is still alive, but currently it’s not eligible to run. In other words.

**> How can Thread go from running to waiting state?**

  By calling **wait()**[method](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

**> How can Thread go from running to sleeping state?**

  By calling **sleep()** [method](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) thread go from running to sleeping state. In sleeping state it will wait for sleep time to get over.

**Terminated (Dead) :** A thread is considered dead **when its run() method completes**.

**You may like to have in depth knowledge of** [**Thread states/ Thread life cycle in java & explanation of thread methods which method puts thread from which state to which state.**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html)

**Question 23. Are you aware of preemptive scheduling and time slicing?**

**Answer.** In **preemptive scheduling**, the **highest priority thread executes until** it enters into the [**waiting or dead state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html).

**In time slicing**, a **thread executes for a certain predefined time** and **then enters runnable pool**. Than thread can enter running state when selected by thread scheduler.

**Question 24. What are** [**daemon threads**](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html)**?**

**Answer.**[Daemon threads](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html) are low priority threads which **runs intermittently in background** for doing **garbage collection**.

   12 Few salient features of [**daemon()** threads](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html)>

* **Thread scheduler schedules these threads** only **when CPU is idle**.
* [Daemon threads](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html) are **service oriented threads**, they **serves all other threads**.
* These threads are **created before user threads are created** and **die after all other user threads dies**.
* **Priority of daemon threads is always 1** (i.e. MIN\_PRIORITY).
* **User created threads are non daemon threads**.
* **JVM can exit** when only daemon threads exist in system.
* we can use **isDaemon()** method to check whether thread is daemon thread or not.
* we can use **setDaemon(boolean on)** method to make any user method a daemon thread.
* If **setDaemon(boolean on)** is called on thread after calling start() method than IllegalThreadStateException is thrown.
* You may like to see how daemon threads work, for that you can use VisualVM or jStack. I have provided Thread dumps over there which shows daemon threads which were intermittently running in background.

Some of the daemon threads which intermittently run in background are >

|  |
| --- |
| "RMI TCP Connection(3)-10.175.2.71" daemon "RMI TCP Connection(idle)" daemon "RMI Scheduler(0)" daemon "C2 CompilerThread1" daemon  "GC task thread#0 (ParallelGC)" |

**Question 25. Why** [**suspend() and resume() methods are deprecated**](http://www.javamadesoeasy.com/2015/03/reason-why-suspend-and-resume-methods.html)**?**

**Answer.**[Suspend()](http://www.javamadesoeasy.com/2015/03/using-suspend-and-resume-method-in.html) method is [**deadlock**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) **prone**. If the target thread holds a lock on object when it is suspended, no thread can lock this object until the target thread is [resumed](http://www.javamadesoeasy.com/2015/03/using-suspend-and-resume-method-in.html). [If the thread that would resume the target thread attempts to lock this monitor prior to calling resume, it results in **deadlock formation**](http://www.javamadesoeasy.com/2015/03/reason-why-suspend-and-resume-methods.html).

These [**deadlocks**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html)are generally called **Frozen processes**.

**Suspend() method puts thread from** [**running to waiting state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html). And thread can go **from waiting to runnable** state **only when resume() method is called** on thread. It is deprecated method.

**Resume()** method is **only used with suspend()** method that’s why it’s also deprecated method.

**Question 26. Why destroy() methods is deprecated?**

**Answer.** This question is again going to check your in depth knowledge of thread methods i.e. [destroy() method](http://www.javamadesoeasy.com/2015/03/destroy-method-in-java-usage-reason-why.html) is [**deadlock**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) **prone**. If the target thread holds a lock on object when it is destroyed, no thread can lock this object (Deadlock formed are similar to deadlock formed when suspend() and resume() methods are used improperly). It results in **deadlock formation**. These [**deadlocks**](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html)are generally called **Frozen processes**.

Additionally you must know calling destroy() method on Threads throw runtimeException i.e. NoSuchMethodError. [**Destroy() method**](http://www.javamadesoeasy.com/2015/03/destroy-method-in-java-usage-reason-why.html) **puts thread from running to** [**dead state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html)**.**

**Question 27. As stop() method is deprecated,  How can we terminate or stop infinitely running thread in java? (Important)**

**Answer.** This is very interesting question where interviewees thread basics basic will be tested. Interviewers tend to know user’s knowledge about main thread’s and thread invoked by main thread.

We will try to address the problem by creating new thread which will run infinitely until certain condition is satisfied and will be called by main Thread.

1. Infinitely running thread can be stopped **using boolean variable.**
2. [Infinitely running thread can be stopped **using interrupt() method**](http://www.javamadesoeasy.com/2015/03/2-alternate-ways-to-stop-thread-as-stop.html)**.**

**Let’s understand Why stop() method is deprecated :**

Stopping a thread with Thread.stop() causes it to release all of the monitors that it has locked. If any of the objects previously protected by these monitors were in an inconsistent state, the damaged objects become visible to other threads, which might lead to unpredictable behavior.

**Question 28. what is significance of yield() method, what state does it put thread in?**

[**yield()**](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) is a **native method** it’s implementation in java 6 has been changed as compared to its implementation java 5. As method is native it’s implementation is provided by JVM.

**In java 5,** yield() method **internally used to call** [**sleep()**](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) **method** giving all the other threads of same or higher priority to execute before yielded thread by leaving allocated CPU for time gap of 15 millisec.

**But java 6**, calling **yield() method gives a hint to the thread scheduler that the current thread is willing to yield its current use of a processor**. The **thread scheduler is free to ignore this hint**. So, sometimes even after using yield() method, you may not notice any difference in output.

salient features of [**yield()**](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) method >

* **Definition** : [yield()](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) **method when called on thread gives a hint to the thread scheduler that the current thread is willing to yield its current use of a processor.The thread scheduler is free to ignore this hint**.
* [**Thread state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **: when** yield() **method is called on thread it goes from running to runnable state**, not in waiting state. Thread is eligible to run but not running and could be picked by scheduler at anytime.
* **Waiting time :** yield() **method stops thread for unpredictable time.**
* **Static method :** yield()is a **static method**, hence calling Thread.yield() causes currently executing thread to yield.
* **Native method :** implementation of yield() method is provided by **JVM**.

Let’s see definition of yield() method as given in java.lang.Thread -

|  |
| --- |
| **publicstaticnativevoid** yield(); |

* [**synchronized block**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) **:** thread **need not to to acquire object lock** before calling yield()method i.e. yield() method **can be called from outside synchronized block**.

**Question 29.What is significance of sleep() method in detail, what** [**state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **does it put thread in ?**

[**sleep()**](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) is a **native method**, its implementation is provided by JVM.

10 salient features of [**sleep()**](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) method >

* **Definition** : [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) **methods causes current thread to sleep for specified number of milliseconds** (i.e. time passed in sleep method as parameter). Ex- Thread.sleep(10) causes currently executing thread to sleep for 10 millisec.
* [**Thread state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **: when** sleep() **is called on thread it goes from running to waiting state** and can return to runnable state when sleep time is up.
* **Exception :** sleep() **method must catch or throw compile time exception** i.e. InterruptedException.
* **Waiting time :** sleep() **method have got few options.**
  1. **sleep(long millis) -** Causes the currently executing thread to sleep for the specified number of milliseconds

|  |
| --- |
| **publicstaticnativevoid** sleep(**long** millis) **throws** InterruptedException; |

* 1. **sleep(long millis, int nanos) -** Causes the currently executing thread to sleep for the specified number of milliseconds plus the specified number of nanoseconds.

|  |
| --- |
| **publicstaticnativevoid** sleep(**long** millis,**int** nanos) **throws** InterruptedException; |

* **static method :** sleep()is a static method, causes the currently executing thread to sleep for the specified number of milliseconds.
* **Native method :** implementation of sleep() method is provided by **JVM**.

Let’s see definition of yield() method as given in java.lang.Thread -

|  |
| --- |
| **publicstaticnativevoid** sleep(**long** millis) **throws** InterruptedException; |

* **Belongs to which class :**[sleep](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html)**[()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) method belongs to java.lang.Thread** class.
* **synchronized block :** thread **need not to to acquire object lock** before calling sleep()method i.e. sleep() method **can be called from outside synchronized block**.

**Question 30. Difference between** [**wait ()**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) **and** [**sleep()**](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) **? (Important)**

**Answer.**

* **Should be called from** [**synchronized block**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) **:**wait() **method is always called from synchronized block** i.e. [wait()](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) method needs to lock object monitor before object on which it is called.  But sleep() **method can be called from outside synchronized block** i.e. sleep() method doesn’t need any object monitor.
* **IllegalMonitorStateException : if** wait() **method is called without acquiring object lock** than IllegalMonitorStateException is thrown at runtime, but sleep() method**never throws such exception**.
* **Belongs to which class : wait() method belongs to java.lang.Object** class but **sleep() method belongs to java.lang.Thread** class.
* **Called on object or thread :** wait() **method is called on objects** but sleep() **method is called on Threads** not objects**.**
* [**Thread state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **: when** wait() **method is called on object, thread that holded object’s monitor goes from running to waiting state** and can **return to runnable state only when notify() or notifyAll()method is called on that object**. And later thread scheduler schedules that thread to go from from runnable to running state.

when sleep() **is called on thread it goes from running to waiting state** and can **return to runnable state when sleep time is up.**

* **When called from** [**synchronized block**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) **:**when wait() method is called **thread leaves the object lock**.  But sleep () method **when called from synchronized block or method thread doesn’t leaves object lock.**

**Question 31. Differences and similarities between yield() and sleep() ?**

**Answer.**

Differences [yield()](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) and [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) :

* **Definition** : yield() **method when called on thread gives a hint to the thread scheduler that the current thread is willing to yield its current use of a processor.The thread scheduler is free to ignore this hint**. sleep() **methods causes current thread to sleep for specified number of milliseconds** (i.e. time passed in sleep method as parameter). Ex- Thread.sleep(10) causes currently executing thread to sleep for 10 millisec.
* [**Thread state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **: when** sleep() **is called on thread it goes from running to waiting state** and can return to runnable state when sleep time is up. **when** yield() **method is called on thread it goes from running to runnable state**, not in waiting state. Thread is eligible to run but not running and could be picked by scheduler at anytime.
* **Exception :** yield() **method need not to catch or throw any exception.** But sleep() **method must catch or throw compile time exception** i.e. InterruptedException.
* **Waiting time :** yield() **method stops thread for unpredictable time, that depends on thread scheduler.** But sleep() **method have got few options.**
  1. **sleep(long millis) -** Causes the currently executing thread to sleep for the specified number of milliseconds
  2. **sleep(long millis, int nanos) -** Causes the currently executing thread to sleep for the specified number of milliseconds plus the specified number of nanoseconds.

similarity between [yield()](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) and [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html):

**>** yield() and sleep() method **belongs to java.lang.Thread** class.

**>**yield() and sleep() method can be **called from outside synchronized block**.

**>** yield() and sleep() method are **called on Threads not objects**.

**Question 32. Mention some g**[**uidelines to write thread safe code, most important point we must take care of in multithreading programs**](http://www.javamadesoeasy.com/2015/03/guidelines-to-thread-safe-code-most.html)**?**

**Answer.**  In multithreading environment it’s important very important to [write thread safe code](http://www.javamadesoeasy.com/2015/03/guidelines-to-thread-safe-code-most.html), thread unsafe code can cause a major threat to your application. I have posted many articles regarding thread safety. So overall this will be revision of what we have learned so far i.e. writing thread safe healthy code and avoiding any kind of [deadlocks](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html).

1. If method is exposed in multithreading environment and it’s not synchronized (thread unsafe) than it might lead us to [race condition](http://www.javamadesoeasy.com/2015/03/race-condition-in-multithreading-and.html), we must try to use [synchronized block and **synchronized methods**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html). [Multiple threads may exist on same object](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_5.html) but only one thread of that object can enter **synchronized method** at a time, though  [threads on different object](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on.html) can enter same method at same time.
2. Even static variables are not thread safe, they are used in static methods and if static methods are not synchronized then thread on same or different object can enter method concurrently. Multiple threads may exist on [same](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_46.html) or [different objects](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on_5.html) of class but only one thread can enter [**static synchronized method**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) at a time, we must consider making [static methods as synchronized](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html).
3. If possible, try to use [**volatile** variables](http://www.javamadesoeasy.com/2015/03/volatile-keyword-in-java-difference.html). If a field is declared volatile all threads see a consistent value for the variable. Volatile variables at times can be used as alternate to synchronized methods as well.
4. **Final variables** are thread safe because once assigned some reference of object they cannot point to reference of other object.

s is pointing to String object.

|  |
| --- |
| **publicclass** MyClass {  **final** String s=**new** String("a");  **void** method(){            s="b"; //compilation error, s cannot point to new reference.     }   } |

If final is holding some primitive value it cannot point to other value.

|  |
| --- |
| **publicclass** MyClass {  **finalint**i=0;  **void** method(){            i=0;  //compilation error, i cannot point to new value.     }   } |

1. Usage of **local variables** : If possible try to use local variables, local variables are thread safe, because every [thread has its own **stack**](http://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html), i.e. every thread has its own local variables and its pushes all the local variables on stack.

|  |
| --- |
| **publicclass** MyClass {  **void** method(){  **int**i=0; //Local variable, is thread safe.     }   } |

1. We must avoid using  **[deadlock prone](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html)** deprecated thread methods such as [destroy()](http://www.javamadesoeasy.com/2015/03/destroy-method-in-java-usage-reason-why.html), [stop()](http://www.javamadesoeasy.com/2015/03/2-alternate-ways-to-stop-thread-as-stop.html), [suspend() and resume()](http://www.javamadesoeasy.com/2015/03/reason-why-suspend-and-resume-methods.html).
2. Using thread safe **collections** : Rather than using ArrayList we must Vector and in place of using HashMap we must use ConcurrentHashMap or HashTable.
3. We must use [VisualVM](http://www.javamadesoeasy.com/2015/03/visualvm-thread-dumps-generating-and_74.html)  or [jstack](http://www.javamadesoeasy.com/2015/03/jstack-thread-dumps-generating-and.html)  to detect problems such as deadlocks and time taken by threads to complete in multi threading programs.
4. Using [*ThreadLocal*](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html):ThreadLocal is a class which provides thread-local variables. Every thread has its own ThreadLocal value that makes ThreadLocal value threadsafe as well.
5. Rather than StringBuffer try using **immutable classes** such as String. Any change to String produces new String.

**Question 33. How thread can enter waiting, sleeping and blocked state and how can they go to runnable state ?**

**Answer.**  This is very prominently asked question in interview which will test your knowledge about [thread states](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html). And it’s very important for developers to have in depth knowledge of this [thread state](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) transition. I will try to explain this thread state transition by framing few sub questions. I hope reading sub questions will be quite interesting.

**> How can Thread go from running to waiting state ?**

  By calling **wait()**[method](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) thread go from running to waiting state. In waiting state it will wait for other threads to release object monitor/lock.

**> How can Thread return from waiting to runnable state ?**

  Once **notify() or notifyAll()**[method](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html) is called object monitor/lock becomes available and thread can again return to runnable state.

**> How can Thread go from running to sleeping state ?**

  By calling **sleep()** [method](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) thread go from running to [sleeping](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) state. In sleeping state it will wait for sleep time to get over.

**> How can Thread return from sleeping to runnable state ?**

  Once specified **sleep time is up** thread can again return to runnable state.

**Suspend()** [method](http://www.javamadesoeasy.com/2015/03/using-suspend-and-resume-method-in.html) can be used to put thread in waiting state and **resume()** method is the only way which could put thread in runnable state.

Thread also may go from running to waiting state if it is waiting for some I/O operation to take place. Once input is available thread may return to running state.

**>**When threads are in running state, **yield()**[method](http://www.javamadesoeasy.com/2015/03/yield-method-in-threads-8-key-features.html) can make thread to go in Runnable state.

**Question 34. Difference between notify() and notifyAll() methods, can you write a code to prove your point?**

**Answer.** Goodness. Theoretically you must have heard or you must be aware of differences between [notify() and notifyAll()](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html).But have you created program to achieve it? If not let’s do it.

First, I will like give you a brief description of what notify() and notifyAll() methods do.

**notify()**- Wakes up a single thread that is [waiting](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is random and occurs at the discretion of the implementation. A thread [waits](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) on an object's monitor by calling one of the wait methods.

[**The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.**](http://www.javamadesoeasy.com/2015/03/the-awakened-threads-will-not-be-able.html)

|  |
| --- |
| **Public final native void** notify(); |

**notifyAll()**- Wakes up all threads that are waiting on this object's monitor. A thread waits on an object's monitor by calling one of the wait methods.

[**The awakened threads will not be able to proceed until the current thread relinquishes the lock on this object.**](http://www.javamadesoeasy.com/2015/03/the-awakened-threads-will-not-be-able.html)

|  |
| --- |
| **publicfinalnativevoid**notifyAll(); |

[Now it’s time to write down a program to prove the point.](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html)

**Question 35. Does thread leaves object lock when** [**sleep()**](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) **method is called?**

**Answer.** When [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) method is called Thread does not leaves object lock and goes from running to waiting state. Thread [waits](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) for sleep time to over and once sleep time is up it goes from [waiting to runnable state](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html).

**Question 36. Does thread leaves object lock when wait() method is called?**

**Answer.** When [wait()](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) method is called Thread leaves the object lock and goes from [running to waiting state](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html). Thread waits for other threads on same object to call notify() or notifyAll() and once any of [notify() or notifyAll()](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html) is called it goes from waiting to runnable state and again acquires object lock.

**Question 37. What will happen if we don’t override run method?**

**Answer.**  This question will test your basic knowledge how start and run methods work internally in Thread Api.

**When we call start() method** on thread, **it internally calls run() method** with newly created thread. **So, if we don’t override run() method newly created thread won’t be called and nothing will happen**.

|  |
| --- |
| **class** MyThread **extends** Thread {     //don't override run() method  }  **publicclass** DontOverrideRun {  **publicstaticvoid** main(String[] args) {            System.*out*.println("main has started.");            MyThread thread1=**new** MyThread();            thread1.start();            System.*out*.println("main has ended.");     }  }  /\*OUTPUT  main has started.  main has ended.  \*/ |

As we saw in output, we didn’t override run() method that’s why on calling start() method nothing happened.

**Question 38. What will happen if we override start method?**

**Answer.** This question will again test your basic core java knowledge how overriding works at runtime, what what will be called at runtime and how start and run methods work internally in Thread Api.

**When we call start() method** on thread, **it internally calls run()** method with newly created thread. **So, if we override start() method, run() method will not be called** until we write code for calling run() method.

|  |
| --- |
| **class** MyThread **extends** Thread {     @Override  **publicvoid** run() {            System.*out*.println("in run() method");     }       @Override  **publicvoid** start(){            System.*out*.println("In start() method");     }    }  **publicclass** OverrideStartMethod {  **publicstaticvoid** main(String[] args) {            System.*out*.println("main has started.");              MyThread thread1=**new** MyThread();            thread1.start();              System.*out*.println("main has ended.");     }  }  /\*OUTPUT  main has started.  In start() method  main has ended.  \*/ |

If we note output. we have overridden start method and didn’t called run() method from it, so, run() method wasn’t call.

**Question 39. Can we acquire lock on class? What are ways in which you can acquire lock on class?**

**Answer.**  **Yes**, we can acquire lock on [class’s class object in 2 ways to acquire lock on class](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html).

Thread can acquire lock on class’s class object by-

1. Entering **synchronized block or**

      Let’s say there is one class MyClass. Now we can create synchronization block, and parameter passed with synchronization tells which class has to be synchronized. In below code, we have synchronized MyClass

**synchronized** (MyClass.class) {

        //thread has acquired lock on MyClass’s class object.

      }

1. by entering **static synchronized methods.**

**public staticsynchronizedvoid** method1() {

        //thread has acquired lock on MyRunnable’s class object.

      }

As soon as thread entered Synchronization method, thread acquired lock on class’s class object.

Thread will leave lock when it exits static synchronized method.

**Question 40. Difference between object lock and class lock?**

**Answer.**  It is very important question from multithreading point of view. We must understand [difference between object lock and class lock](http://www.javamadesoeasy.com/2015/03/difference-between-object-lock-and.html) to answer interview, ocjp answers correctly.

|  |  |
| --- | --- |
| [**Object lock**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) | [**Class lock**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) |
| Thread can acquire [object lock](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) by-   1. Entering **synchronized block or** 2. by entering **synchronized methods.** | Thread can acquire lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) by-   1. Entering **synchronized block or** 2. by entering **static synchronized methods.** |
| [Multiple threads may exist on same object but only one thread of that object can enter **synchronized method** at a time.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_5.html)  [Threads on different object can enter same method at same time.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on.html) | Multiple threads may exist on [same](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_46.html) or [different objects](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on_5.html) of class but only one thread can enter **static synchronized method** at a time. |
| **Multiple objects of class may exist and every object has it’s own lock.** | **Multiple objects of class may exist but there is always one class’s class object lock available**. |
| First let’s acquire [object lock](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) by entering **synchronized block**.  Example- Let’s say there is one class MyClassand we have created it’s object and reference to that object is myClass. Now we can create synchronization block, and parameter passed with synchronization tells which object has to be synchronized. In below code, we have synchronized object reference by myClass.  MyClass myClass=**new**Myclass();  **synchronized** (myClass) {       }  As soon thread entered Synchronization block, thread acquired object lock on object referenced by myClass (by acquiring object’s monitor.)  Thread will leave lock when it exits synchronized block. | First let’s acquire lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) by entering **synchronized block.**  Example- Let’s say there is one class MyClass. Now we can create synchronization block, and parameter passed with synchronization tells which class has to be synchronized. In below code, we have synchronized MyClass  **synchronized** (MyClass.class) {     }  As soon as thread entered Synchronization block, thread acquired MyClass’s class object. Thread will leave lock when it exits synchronized block. |
| **publicsynchronizedvoid** method1() {  }  As soon as thread entered **Synchronization method**, thread acquired [object lock](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html).  Thread will leave lock when it exits synchronized method. | **public staticsynchronizedvoid**method1() {}  As soon as thread entered **static Synchronization method**, thread acquired lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html).  Thread will leave lock when it exits synchronized method. |

**Let’s me give you some tricky situation based question,**

**Question 41.** Suppose you have **2 threads (Thread-1 and Thread-2) on same object**. **Thread-1** is in **synchronized method1(),** can **Thread-2** enter **synchronized method2()** at same time?

**Answer.No**, here when Thread-1 is in **synchronized method1()** it must be **holding** [**lock on object’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) and will release lock on object’s monitor only when it exits **synchronized method1()**. So, Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) for Thread-1 to release lock on object’s monitor so that it could enter **synchronized method2()**.

**Likewise**, Thread-2 even cannot enter **synchronized method1()** which is being executed by Thread-1. Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) for Thread-1 to release lock on object’s monitor so that it could enter **synchronized method1()**. [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_5.html)

**Question 42.** Suppose you have **2 threads (Thread-1 and Thread-2) on same object**. **Thread-1** is in **static synchronized method1(),** can **Thread-2** enter **static synchronized method2()** at same time?

**Answer.No**, here when Thread-1 is in **static synchronized method1()** it must be **holding lock on** [**class class’s object**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) and will release lock on class’s classobject only when it exits **static synchronized method1()**. So, Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) for Thread-1 to release lock on class’s classobject so that it could enter **static synchronized method2()**.

**Likewise**, Thread-2 even cannot enter **static synchronized method1()** which is being executed by Thread-1. Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) for Thread-1 to release lock on  class’s classobject so that it could enter **static synchronized method1()**. [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_46.html)

**Question 43.** Suppose you have **2 threads (Thread-1 and Thread-2) on same object**. **Thread-1** is in **synchronized method1(),** can **Thread-2** enter **static synchronized method2()** at same time?

**Answer.Yes**, here when Thread-1 is in **synchronized method1()** it must be **holding** [**lock on object’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) and Thread-2 can enter **static synchronized method2()** by acquiring lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html). [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-and_65.html)

**Question 44.** Suppose you have thread and it is in **synchronized method** and now can thread **enter other synchronized method** from that method?

**Answer.Yes**, here when thread is in **synchronized method** it must be **holding** [**lock on object’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) and **using that lock** thread can **enter other synchronized method**. [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-thread-and-it-is-in_5.html)

**Question 45.** Suppose you have thread and it is in **static synchronized method** and now can thread **enter other static synchronized method** from that method?

**Answer.**  **Yes**, here when thread is in **static synchronized method** it must be **holding lock on** [**class’s class object**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) and **using that lock** thread can **enter other static synchronized method**. [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-thread-and-it-is-in_16.html)

**Question 46.** Suppose you have thread and it is in **static synchronized method** and now can thread **enter other non static synchronized method** from that method?

**Answer.Yes**, here when thread is in **static synchronized method** it must be **holding lock on** [**class’s class object**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) and when it **enters synchronized method** it will **hold** [**lock on object’s monitor**](http://v/) **as well**.

So, now thread holds 2 locks (it’s also called nested synchronization)-

**>**first one on **class’s class object.**

**>**second one on **object’s monitor** (This lock will be released when thread exits non static method)**.**[Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-thread-and-it-is-in_41.html)

**Question 47.** Suppose you have thread and it is in **synchronized method** and now can thread **enter other static synchronized method** from that method?

**Answer.Yes**, here when thread is in synchronized method it must be holding [**lock on object’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html) and when it enters static synchronized method it will hold lock on [class’s class object](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) as well.

So, now thread holds 2 locks (it’s also called nested synchronization)-

**>**first one on [**object’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html)**.**

**>**second one on **class’s class object.**(This lock will be released when thread exits static method)**.**[Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-thread-and-it-is-in_17.html)

**Question 48.** Suppose you have **2 threads (Thread-1 on object1 and Thread-2 on object2)**. **Thread-1** is in **synchronized method1(),** can **Thread-2** enter **synchronized method2()** at same time?

**Answer.Yes**, here when Thread-1 is in **synchronized method1()** it must be **holding** [**lock on object1’s monitor**](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html). Thread-2 will acquire lock on **object2’s monitor** and enter **synchronized method2()**.

**Likewise**, Thread-2 even enter **synchronized method1()** as well which is being executed by Thread-1 (because threads are created on different objects). [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on.html)

**Question 49.** Suppose you have **2 threads (Thread-1 on object1 and Thread-2 on object2)**. **Thread-1** is in **static synchronized method1(),** can **Thread-2** enter **static synchronized method2()** at same time?

**Answer.No**, it might confuse you a bit that threads are created on different objects. But, not to forgot that **multiple objects may exist but there is always one** [**class’s class object**](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) **lock available**.

Here, when Thread-1 is in **static synchronized method1()** it must be **holding lock on class class’s object** and will release lock on class’s classobject only when it exits **static synchronized method1()**. So, Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) for Thread-1 to release lock on class’s classobject so that it could enter **static synchronized method2()**.

**Likewise**, Thread-2 even cannot enter **static synchronized method1()** which is being executed by Thread-1. Thread-2 will have to [wait](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) for Thread-1 to release lock on  [class’s classobject](http://www.javamadesoeasy.com/2015/03/acquiring-lock-on-class-2-ways-to.html) so that it could enter **static synchronized method1()**. [Now, let’s see a program to prove our point.](http://www.javamadesoeasy.com/2015/03/suppose-you-have-2-threads-thread-1-on_5.html)

**Question 50. Difference between wait() and wait(long timeout), What are** [**thread states**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html) **when these method are called?**

**Answer.**

|  |  |
| --- | --- |
| [**wait()**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) | **wait(long timeout)** |
| When [wait()](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) method is called on object, it causes causes the current thread to wait until another thread invokes the notify() or notifyAll() method for this object. | **wait(long timeout) -** Causes the current thread to wait until either another thread invokes the notify() or notifyAll() methods for this object, or a specified timeout time has elapsed. |
| **When** [**wait()**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) **is called** on object - Thread enters from [**running to waiting state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html).  **It** [**waits**](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) **for some other thread to call notify so that it could enter runnable state**. | **When wait(1000) is called** on object - Thread enters from **running to waiting state**. Than **even if notify() or notifyAll() is not called after  timeout time has elapsed thread will go from** [**waiting to runnable state**](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html)**.** |

**Question 51.  How can you implement your own Thread Pool in java?**

**Answer.**

*What is* [*ThreadPool*](http://www.javamadesoeasy.com/2015/03/implement-thread-pool-in-java.html)*?*

ThreadPool is a pool of threads which **reuses a fixed number of threads**  to execute tasks.

At any point, **at most nThreads threads will be active processing tasks**. **If additional tasks are submitted when all threads are active, they will wait in the queue until a thread is available**.

ThreadPool implementation internally uses [LinkedBlockingQueue](http://www.javamadesoeasy.com/2015/03/custom-implementation-of.html) for adding and removing tasks.

In this post i will be using LinkedBlockingQueue provide by java Api, you can refer this post for [implementing ThreadPool using custom LinkedBlockingQueue](http://www.javamadesoeasy.com/2015/03/implementing-threadpool-using-custom.html).

*Need/Advantage of ThreadPool?*

**Instead of creating new thread every time for executing tasks**, we can create **ThreadPool** which **reuses a fixed number of threads for executing tasks**.

As threads are reused, performance of our application improves drastically.

*How ThreadPool works?*

We will instantiate ThreadPool, in ThreadPool’s **constructor** nThreads number of threads are created and started.

|  |
| --- |
| ThreadPool threadPool=**new** ThreadPool(2); |

Here 2 threads will be created and started in ThreadPool.

Then, threads will enter **run()** method of **ThreadPoolsThread** class and will call take() method on taskQueue.

* If tasks are available thread will execute task by entering run() method of task (As tasks executed always implements Runnable).

|  |
| --- |
| **publicvoid** run() {  . . .  **while** (**true**) {          . . .  **Runnable runnable = taskQueue.take();**  **runnable.run();**          . . .      }  . . .  } |

* Else waits for tasks to become available.

**When tasks are added?**

When execute() method of **ThreadPool** is called, it internally calls put() method on taskQueue to add tasks.

|  |
| --- |
| taskQueue.put(task); |

Once tasks are available all waiting threads are notified that task is available.

More detail on how to [Implement Thread pool in java](http://www.javamadesoeasy.com/2015/03/implement-thread-pool-in-java.html).

**Question 52.  What is significance of using** [**ThreadLocal**](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html)**?**

**Answer.**  This question will test your command in multi threading, can you really create some perfect multithreading application or not. [ThreadLocal](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html) is a class which provides thread-local variables.

*What is* [*ThreadLocal*](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html) *?*

ThreadLocal is a class which provides thread-local variables. Every thread has its own ThreadLocal value that makes ThreadLocal value threadsafe as well.

*For how long Thread holds ThreadLocal value?*

Thread holds ThreadLocal value till it hasn’t entered [dead state](http://www.javamadesoeasy.com/2015/03/thread-states-thread-life-cycle-in-java.html).

*Can one thread see other thread’s ThreadLocal value?*

**No**, thread can see only it’s ThreadLocal value.

*Are ThreadLocal variables thread safe. Why?*

**Yes**, ThreadLocal variables are thread safe. As every thread has its own ThreadLocal value and one thread can’t see other threads ThreadLocal value.

*Application of* [*ThreadLocal*](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html)*?*

1. ThreadLocal are **used by many web frameworks** for maintaining some context (may be session or request) related value.
   * In any **single threaded application**, same thread is assigned for every request made to same action, so ThreadLocal values will be available in next request as well.
   * In **multi threaded application**, different thread is assigned for every request made to same action, so ThreadLocal values will be different for every request.
2. When threads have started at different time they might like to store time at which they have started. **So, thread’s start time can be stored in ThreadLocal.**

*Creating ThreadLocal >*

|  |
| --- |
| **private** ThreadLocal<String>threadLocal =   **new** ThreadLocal<String>(); |

We will create instance of ThreadLocal. ThreadLocal is a generic class, i will be using String to demonstrate threadLocal.

**All threads will see same instance of ThreadLocal**, **but a thread will be able to see value which was set by it only**.

*How thread set value of ThreadLocal >*

|  |
| --- |
| threadLocal.set( **new** Date().toString()); |

Thread set value of ThreadLocal by calling set(“”) method on threadLocal.

*How thread get value of ThreadLocal >*

|  |
| --- |
| threadLocal.get() |

Thread get value of ThreadLocal by calling get() method on threadLocal.

See here for detailed explanation of [threadLocal](http://www.javamadesoeasy.com/2015/03/threadlocal-in-multithreading-in-java.html).

**Question 53. What is busy spin?**

**Answer.**

*What is* [*busy spin*](http://www.javamadesoeasy.com/2015/03/busy-spin-what-is-busy-spin-consumer.html)*?*

When one thread loops continuously waiting for another thread to signal.

*Performance point of view* - Busy spin is **very bad** from performance point of view, because one thread keeps on looping continuously ( and consumes CPU) waiting for another thread to signal.

*Solution to busy spin -*

We must use [sleep()](http://www.javamadesoeasy.com/2015/03/sleep-method-in-threads-10-key-features.html) or [wait() and notify()](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) method. Using wait() is better option.

*Why using wait() and notify() is much better option to solve busy spin?*

Because in case when we use sleep() method, thread will wake up again and again after specified sleep time until boolean variable is true. But, in case of wait() thread will wake up only when when notified by calling [notify() or notifyAll()](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html), hence end up consuming CPU in best possible manner.

*Program - Consumer Producer problem with busy spin >*

Consumer thread continuously execute (**busy spin**) in while loop till**productionInProcess** is true. Once producer thread has ended it will make boolean variable **productionInProcess** false and **busy spin** will be over.

|  |
| --- |
| **while(productionInProcess){**  **System.*out*.println("BUSY SPIN - Consumer waiting for production to get over");**  **}** |

[See here for Busy spin in detail](http://www.javamadesoeasy.com/2015/03/busy-spin-what-is-busy-spin-consumer.html).

**Question 54. Can a constructor be synchronized?**

**Answer.**  No, constructor cannot be synchronized. Because constructor is used for instantiating object, when we are in constructor object is under creation. So, until object is not instantiated it does not need any synchronization.

**Enclosing** constructor in synchronized block will generate compilation error.

Using synchronized in **constructor definition** will also show compilation error.

COMPILATION ERROR = Illegal modifier for the constructor in type ConstructorSynchronizeTest; only public, protected & private are permitted

**Though we can use synchronized block inside constructor.**

*Read More about :* [***Constructor in java cannot be synchronized***](http://www.javamadesoeasy.com/2015/03/constructor-in-java-cannot-be.html)

**Question 55. Can you find whether thread holds lock on object or not?**

**Answer.**  holdsLock(object) method can be used to find out whether current thread holds the lock on monitor of specified object.

holdsLock(object) method returns true if the current thread holds the lock on monitor of specified object.

**Question 56. What do you mean by thread starvation?**

**Answer.**   When thread does not enough CPU for its execution **Thread starvation happens.**

**Thread starvation** may happen in following scenarios >

* Low priority threads gets less CPU (time for execution) as compared to high priority threads. **Lower priority thread** may **starve** away waiting to get enough CPU to perform calculations.
* In [deadlock](http://www.javamadesoeasy.com/2015/03/deadlock-in-multithreading-program-to.html) two threads waits for each other to release lock holded by them on resources. There both **Threads starves away to get CPU.**
* Thread might be waiting indefinitely for lock on object’s monitor (by calling [wait()](http://www.javamadesoeasy.com/2015/03/wait-and-notify-methods-definition-8.html) method), because no other thread is calling [notify()/notifAll()](http://www.javamadesoeasy.com/2015/03/difference-between-notify-and-notifyall.html) method on object. In that case, **Thread starves** away to get CPU.
* Thread might be waiting indefinitely for lock on object’s monitor (by calling wait() method), but notify() may be repeatedly awakening some other threads. In that case also **Thread starves** away to get CPU.

**Question 57. What is addShutdownHook method in java?**

**Answer.**  [addShutdownHook](http://www.javamadesoeasy.com/2015/03/threads-addshutdownhook-method-in-java.html) method in java >

* addShutdownHook method **registers a new virtual-machine shutdown hook**.
* A shutdown hook is a **initialized but unstarted thread**.
* When **JVM starts its shutdown** it will **start all registered shutdown hooks** in some unspecified order and let them run concurrently.

When JVM (Java virtual machine)  shuts down >

* When the last non-[daemon](http://www.javamadesoeasy.com/2015/03/daemon-threads-12-salient-features-of.html) thread finishes, or
* when the System.exit is called.

*Once JVM’s shutdown has begun***new shutdown hook cannot be registered** neither  **previously-registered hook can be de-registered**. Any attempt made to do any of these operations causes an IllegalStateException.

*For more detail with program read :* [***Threads addShutdownHook method in java***](http://www.javamadesoeasy.com/2015/03/threads-addshutdownhook-method-in-java.html)

**Question 58. How you can handle uncaught runtime exception generated in run method?**

**Answer.**  We can use [setDefaultUncaughtExceptionHandler](http://www.javamadesoeasy.com/2015/03/handling-uncaught-runtime-exception.html) method which can handle uncaught unchecked(runtime) exception generated in run() method.

What is setDefaultUncaughtExceptionHandler method?

setDefaultUncaughtExceptionHandler method sets the default handler which is called when a thread terminates due to an uncaught unchecked(runtime) exception.

*setDefaultUncaughtExceptionHandler method features >*

* **setDefaultUncaughtExceptionHandler** method sets the default handler which is called when a thread terminates due to an uncaught unchecked(runtime) exception.
* **setDefaultUncaughtExceptionHandler** is a static method method, so we can directly call  Thread.***setDefaultUncaughtExceptionHandler*** to set the default handler to handle uncaught unchecked(runtime) exception.
* It avoids abrupt termination of thread caused by uncaught runtime exceptions.

Defining setDefaultUncaughtExceptionHandler method >

|  |
| --- |
| Thread.***setDefaultUncaughtExceptionHandler***(**new** Thread.UncaughtExceptionHandler(){  **publicvoid** uncaughtException(Thread thread, Throwable throwable) {     System.*out*.println(thread.getName() + " has thrown " + throwable);     }    }); |

*For more detail read :* [*Program to demonstrate* ***setDefaultUncaughtExceptionHandler*** *method.*](http://www.javamadesoeasy.com/2015/03/handling-uncaught-runtime-exception.html)

**Question 59. What is ThreadGroup in java, What is default priority of newly created threadGroup, mention some important ThreadGroup methods ?**

**Answer.**  When program starts **JVM creates  a ThreadGroup** named ***main***. Unless specified, all  newly created threads become members of the ***main*** thread group.

**ThreadGroup is initialized with default priority of 10.**

*ThreadGroup* ***important methods*** *>*

* **getName()** 
  + name of ThreadGroup.
* **activeGroupCount()**
  + count of active groups in ThreadGroup.
* **activeCount()**
  + count of active threads in ThreadGroup.
* **list()**
  + list() method has prints ThreadGroups information
* **getMaxPriority()**
  + Method returns the maximum priority of ThreadGroup.
* **setMaxPriority(int pri)**
  + Sets the maximum priority of ThreadGroup.

*Read more about* [***ThreadGroup in java***](http://www.javamadesoeasy.com/2015/03/threadgroup-in-java.html)*.*

**Question 60. What are thread priorities?**

**Answer.**

[*Thread Priority*](http://www.javamadesoeasy.com/2015/03/thread-priorities-setpriority-and.html) *range is from 1 to 10.*

Where **1 is minimum priority** and **10 is maximum priority.**

Thread class provides variables of **final static int** type for setting thread priority.

|  |
| --- |
| /\* The minimum priority that a thread can have. \*/  **publicfinalstaticint*MIN\_PRIORITY***= 1;      /\* The default priority that is assigned to a thread. \*/  **publicfinalstaticint*NORM\_PRIORITY***= 5;     /\* The maximum priority that a thread can have. \*/  **publicfinalstaticint*MAX\_PRIORITY***= 10; |

Thread with **MAX\_PRIORITY is likely to get more CPU** as compared to low priority threads. But **occasionally low priority thread might get more CPU**. Because thread scheduler schedules thread on discretion of implementation and [thread behaviour is totally unpredictable](http://www.javamadesoeasy.com/2015/03/thread-behaviour-is-unpredictable.html).

Thread with **MIN\_PRIORITY is likely to get less CPU** as compared to high priority threads. But **occasionally high priority thread might less CPU**. Because thread scheduler schedules thread on discretion of implementation and thread behaviour is totally unpredictable.

***setPriority()* method is used for Changing the priority of thread.**

***getPriority()* method returns the thread’s priority.**

**Collection interview Question 1. What is Collection framework in java?**

**Answer**. It’s the basic Collection framework interview question. Freshers must know about this.  [***java.util.Collection***](http://www.javamadesoeasy.com/2015/04/collection-in-java.html)*is the* root interface in the *hierarchy of Java Collection framework in java*.

The JDK does not provide any classes which directly implements this interface, but it provides classes which are implementations of more specific subinterfaces like [Set and List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) in java.

java.util.**[Set](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html)** extends java.util.Collection interface in java.

[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

java.util.**[List](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)** extends java.util.Collection interface in java.

[**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

***Also read >***

[COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-2 > Q51- Q75](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview.html)

[**COLLECTION - Top 100 important interview OUTPUT questions and answers in java, Set-3 > Q75- Q100**](http://www.javamadesoeasy.com/2015/07/collection-top-100-important-interview45.html)

**Collection interview Question 2. Which interfaces and classes are most frequently used in Collection framework in java?**

**Answer**. This collection framework interview question will test your practical knowledge. Freshers may get away by answering few interface and classes but experienced developers must answer this question in detail.

**Most frequently used interface in Collection framework are >**

[**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html), [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) and [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html).

**Most frequently used classes in Collection framework are >**

[**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

[**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

[**HashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)***,*** [**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)***,*** [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)***,*** [**LinkedHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), [**TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html), [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) classes implements **Map** interface.

**Collection interview Question 3. What are subinterfaces of Collection interface in java? Is Map interface also a subinterface of Collection interface in java?**

**Answer**. [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) and [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) are subinterfaces of java.util.[**Collection**](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) in java.

*It’s important to note* [***Map***](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) *interface is a member of the Java Collections Framework, but it does not implement Collection interface in java.*

**Collection interview Question 4. What are differences between** [**ArrayList and LinkedList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html) **in java?**

**Answer**. This is very important collection framework interview question in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | **java.util.LinkedList** |
| 1 | Structure | java.util.ArrayList is index based structure in java.  [https://lh3.googleusercontent.com/wCfo_q1uxCzZZCgGpetqEQYMeVj9YMJokT9-WJ7QY4jxCF11u5-WVIjVheBCfKlPJtQ9Bp5zzxTJcPgYLMr0N3n6PvjXPzd-7O-FJr2KoW7qrUjERB-yXK2YxFkH6qrLAX6hvdg5](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | A java.util.**LinkedList** is a data structure consisting of a group of **nodes** which together represent a sequence.  node is composed of a data and a reference (in other words, a **link**) to the next node in the sequence in java.  [https://lh3.googleusercontent.com/ykSE04usYkDTj50vuGVTWKtVGJootTOKa07Eub-E6D5KkOCNAb399G4agtbSKOyeaPAUvAngY6JjDMs-SBNmblDOXLv62eHNVIwEuGD5-GNXTP45Ubtyp0BYg0seOxGSpXHatWJP](http://www.javamadesoeasy.com/2015/01/doublylinkedlist-insert-and-delete-at.html) |
| 2 | **Resizable** | **ArrayList is Resizable-array in java.** | New node is created for storing new element in LinkedList in java. |
| 3 | **Initial capacity** | java.util.ArrayList is created with initial capacity of 10 in java. | For storing every element node is created in LinkedList, so linkedList’s initial capacity is 0 in java. |
| 4 | Ensuring **Capacity**/ resizing. | ArrayList is created with initial capacity of 10.  ArrayList’s size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | For storing every element node is created, so linkedList’s initial capacity is 0, it’s size grow with addition of each and every element in java. |
| 5 | RandomAccess interface | ArrayList implements RandomAccess(Marker interface) to indicate that they support fast random access (i.e. index based access) in java. | LinkedList does not implement RandomAccess interface in java. |
| 6 | AbstractList and AbstractSequentialList | ArrayList extends AbstractList (abstract class) which provides implementation to  List interface to minimize the effort required to implement this interface backed by RandomAccess interface. | LinkedList extends AbstractSequentialList (abstract class), AbstractSequentialList extends AbstractList.  In LinkedList, data is accessed sequentially, so for obtaining data at specific index, iteration is done on nodes sequentially in java. |
| 7 | How **get(index)** method works?  (Though difference has been discussed briefly in above 2 points but in this in point we will figure difference in detail.) | Get method of ArrayList directly gets element on specified index. Hence, offering O(1) complexity in java. | Get method of LinkedList iterates on nodes sequentially to get element on specified index. Hence, offering O(n) complexity in java. |
| **8** | **When to use** | **Use ArrayList when get operations is more frequent than add and remove operations in java.** | **Use LinkedList when add and remove operations are more frequent than get operations in java.** |

For more detail like complexity comparison of method please read : [**ArrayList vs LinkedList in java**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-linkedlist-similarity-and.html)

**Collection interview Question 5. What are differences between** [**ArrayList and Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html) **in java?**

**Answer**. Another very important collection framework interview question to differentiate between ArrayList and Vector in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | ***java.util.Vector*** |
| 1 | synchronization | java.util.ArrayList is **not synchronized**  (because 2 threads on same ArrayList object can access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/consequence-of-using-arraylist-in.html)to show consequence of using ArrayList in multithreading environment.  In the program we will implement our own arrayList in java. | java.util.Vector is **synchronized** (because 2 threads on same Vector object cannot  access it at same time).  I have created [**program**](http://www.javamadesoeasy.com/2015/05/advantage-of-using-vector-in.html)to show advantage of using Vector in multithreading environment.  In the program we will implement our own vector in java. |
| 2 | Performance | ArrayList is not synchronized, hence its operations are **faster** as compared to Vector in java. | Vector is synchronized, hence its operations are **slower** as compared to ArrayList in java.  If we are working not working in multithreading environment jdk recommends us to use ArrayList. |
| 3 | Enumeration | **Enumeration** is [**fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), means any modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException in java. | **Enumeration** is **fail-safe**, means any modification made to Vector during iteration using Enumeration don’t throw any exception in java. |
| 4 | Introduced  in which java version | ArrayList was introduced in second version of java i.e. **JDK 2.0** | Vector was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the List interface, hence making it a member of member of the[Java Collections Framework](http://www.javamadesoeasy.com/2015/04/collection-in-java.html). |
| 5 | Ensuring Capacity/ resizing. | ArrayList is created with initial capacity of 10.  When its full size is **increased by 50%** i.e. after resizing it’s size become 15 in java. | Vector is created with initial capacity of 10.  Vector’s size is **increased by 100%** i.e. after resizing it’s size become 20 in java. |
| 6 | Custom implementation | [https://lh3.googleusercontent.com/2yHNtovknpsdxOKpK4Sd3oFiHP3fKhhrMsZDH3DJaRNsvWB7RnEqtXjyS5yrk6175OwELqF6-viscZQxK8uMK58-gmsz1tN0sHmVSJBEwKJ1UZwle61DItNZeF8MDwyFx-NUXHDU](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)Read : [ArrayList custom implementation](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html) | [https://lh5.googleusercontent.com/Gm_SkaJR6TXlRpDt3ipjw739Gfcg4b2V-pMT4WWrw0cMVTsUhDvApjrvMATYF4XiBotrk8O0Sbc7kwi_v0V7SrrY_cfTUU5dR0_rEogBBG34UB1IviiWwIHcpc5XuH_k5KmOsnmP](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html)Read :  [Vector custom implementation](http://javamadesoeasy.com/2015/02/vector-custom-implementation.html) |

For more detail like complexity comparison of method please read: [**ArrayList vs Vector- Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)

**Collection interview Question 6. What are differences between** [**List and Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) **interface in java?**

**Answer**. Another very very important collection framework interview question to differentiate between **List and Set** in java.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.List*** | ***java.util.Set*** |
| 1 | Insertion order | java.util.List is ordered collection it **maintain insertion order** in java. | *Most of the java.util.Set implementation* does not **maintain insertion order**.  HashSet does not maintains insertion order in java.  Thought LinkedHashSet maintains insertion order in java.    TreeSet is sorted by natural order in java. |
| 2 | Duplicate elements | List **allows to store duplicate elements** in java. | *Set does* ***not allow to store duplicate elements*** in java*.* |
| 3 | Null keys | List allows to store **many null keys** in java. | Most of the Set implementations allow to add only **one null** in java**.**  TreeSet does not allow to add null in java. |
| 4 | Getting element on specific **index** | List implementations provide get method to get element on specific index in java.  ArrayList, Vector, copyOnWriteArrayList and LinkedList provides -  *get(int index)*  Method returns element on specified *index*.  **Get method directly gets element on specified index. Hence, offering O(1) complexity.** | Set implementations does not provide any such get method to get element on specified index in java. |
| 5 | Implementing classes | [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***,*** [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes implements [**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. | [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)***,*** [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***,*** [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)***,*** [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) classes implements [**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface in java. |
| 6 | listIterator | **listIterator** method returns listIterator to iterate over elements in List in java.  **listIterator provides** additional methods as compared to iterator like  **hasPrevious(), previous(), nextIndex(), previousIndex(), add(E element), set(E element)** | Set does not provide anything like listIterator. It simply return Iterator in java. |
| 7 | Structure and resizable | **List** are Resizable-array implementation of the java.util.**List** interface in java. | Set uses [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)for their implementation.  Hence, structure is map based and resizing depends on Map implementation.  *Example >*[***HashSet***](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *internally uses* [*HashMap*](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)*.* |
| 8 | Index based structure /RandomAccess | As **ArrayList** uses array for implementation it is index based structure, hence provides random access to elements.  But [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) is not indexed based structure in java. | Set is not index based structure at all in java. |

For more detail read : [**List vs Set - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html)

**Collection interview Question 7. What are differences between** [**Iterator and ListIterator**](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html)**? in java**

**Answer**. This collection framework interview question is tests your knowledge of iterating over different collection framework classes in java.

|  |  |  |
| --- | --- | --- |
|  | ***java.util.ListIterator*** | ***java.util.Iterator*** |
| 1 | **hasPrevious()**  method returns true if this listIterator has more elements when traversing the list in the reverse direction. | **No such method** in java.util.Iterator. |
| 2 | **previous()**  returns previous element in iteration (traversing in backward direction).  if the iteration has no previous elements than NoSuchElementException is thrown. | **No such method** in java.util.Iterator. |
| 3 | **nextIndex()**  method returns the index of the element that would be returned by a subsequent call to next() method. If listIterator is at the end of the list than method returns size of list. | **No such method** in java.util.Iterator. |
| 4 | **previousIndex()**  method returns the index of the element that would be returned by a subsequent call to previous() method. If listIterator is at the start of the list than method returns -1. | **No such method** in java.util.Iterator. |
| 5 | **add(E element)**  Method inserts the specified **element** into the list.  The element is inserted immediately before the element that would be returned by next (So, subsequent call to next would be unaffected), if any, and after the element that would be returned by previous (So,subsequent call to previous would return the new**element**), if any.  If the list does not contain any element than new **element** will be the sole element in the list. | **No such method** in java.util.Iterator. |
| 6 | **set(E element)**  Method replaces the last element returned by next() or previous() method with the specified **element**. This call can be made only if neither remove nor add have been called after the last call to next or previous.  If call to set() method is followed up by any call made to remove() or add() method after next() or previous() than UnsupportedOperationException is thrown. | **No such method** in java.util.Iterator. |
| 7 | All the implementations of [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) interface like [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)***,*** [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)***,*** [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) classes returns listIterator. | All Implementation classes of [**Collection**](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) interface’s subinterfaces like [Set and List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) return iterator. |

For more detail read : [**Iterator vs ListIterator - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/iterator-vs-listiterator-similarity-and.html)

**Collection interview Question 8. What are differences between** [**Collection and Collections**](http://www.javamadesoeasy.com/2015/04/collection-vs-collections-differences.html) **in java?**

**Answer**.  This is another very important collection framework interview question.In real projects you must have used both Collection and Collections but what is the difference between two of them in java?

java.util.***[Collection](http://www.javamadesoeasy.com/2015/04/collection-in-java.html) ​*** *is the* root **interface** in the ​*hierarchy of Java Collection framework​*.

The JDK does not provide any classes which directly implements java.util.Collection interface, but it  provides classes such as [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html), [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html), [**vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html), [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html), [**EnumSet**](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html), [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html), [CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html), [CopyOnWriteArraySet](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html), [ConcurrentSkipListSet](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)  which implements more specific subinterfaces like ​[Set and List​](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) in java.

java.util.**Collections** is a utility **class** which **consists** of **static methods** that **operate on** or return **Collection** in java.

**java.util.Collections provides method like >**

* **reverse** method for reversing [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) in java.
* **shuffle** method for shuffling elements of **List** in java.
* **unmodifiableCollection**, [**unmodifiableSet**](http://www.javamadesoeasy.com/2015/04/hashset-making-set-unmodifiable-using.html), [**unmodifiableList**](http://www.javamadesoeasy.com/2015/04/arraylist-making-list-unmodifiable.html), [**unmodifiableMap**](http://www.javamadesoeasy.com/2015/04/hashmap-making-map-unmodifiable-using.html) methods for making **List**, [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) and [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) unmodifiable in java.
* **min** method to return smallest element in **Collection** in java.
* **max** method to return smallest element in **Collection**.
* **sort** method for sorting **List**.
* **synchronizedCollection**, [**synchronizedSet**](http://www.javamadesoeasy.com/2015/04/hashset-synchronizing-using.html), [**synchronizedList**](http://www.javamadesoeasy.com/2015/04/arraylist-synchronizing-using.html), [**synchronizedMap**](http://www.javamadesoeasy.com/2015/04/hashmap-synchronizing-map-using.html)methods for synchronizing **List**, **Set** and **Map** respectively in java**.**

Additionally you must know that *java.util.Collection and java.util.Collections both were introduced in* ***second version of java i.e. in JDK 2.0.***

**Collection interview Question 9. What are core classes and interfaces in java.util.List hierarchy in java?**

**Answer**. Freshers must know core classes in List hierarchy but experienced developers must be able to explain this java.util.List hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html)

java.util.**[List](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html)** interface extends java.util.Collection interface.

java.util.**[ArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)*, java.util.***[**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)***, java.util.*[Vector](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)*, java.util.concurrent.*[CopyOnWriteArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)**classes implements java.util.[**List**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

Also some abstract classes like java.util.**AbstractCollection**, java.util.**AbstractList** and java.util.**AbstractSequentialList** have been mentioned in hierarchy.

**Collection interview Question 10. What are core classes and interfaces in java.util.Set hierarchy?**

**Answer**. Freshers must know core classes in Set hierarchy but experienced developers must be able to explain this java.util.Set hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)

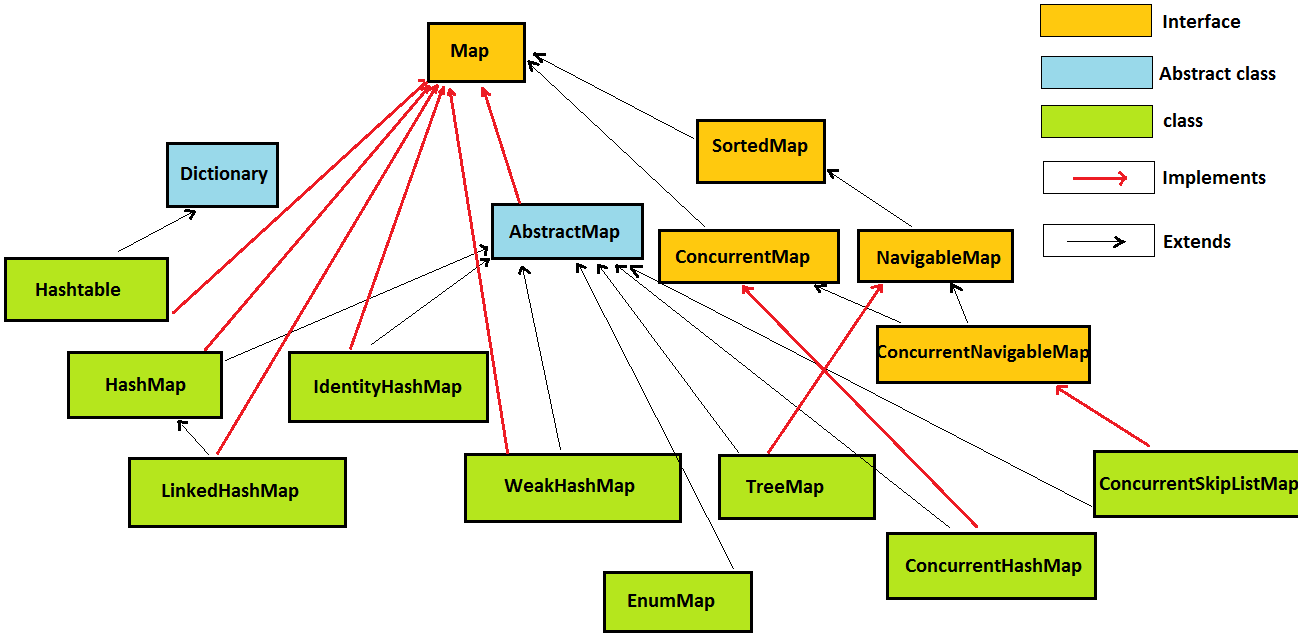
java.util.**[Set](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html)** interface extends java.util.Collection interface.

java.util.**[HashSet](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*, java.util.concurrent.***[**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)***, java.util.*[LinkedHashSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)*, java.util.*[TreeSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)**, java.util.concurrent.**[ConcurrentSkipListSet](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)**, java.util.**[EnumSet](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html)** classes implements java.util.[**Set**](http://www.javamadesoeasy.com/2015/04/list-vs-set-similarity-and-differences.html) interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractSet** and java.util.**AbstractCollection** have been mentioned in hierarchy.

**Collection interview Question 11. What are core classes and interfaces in java.util.Map hierarchy?**

**Answer**. Freshers must know core classes in Map hierarchy but experienced developers must be able to explain this java.util.Map hierarchy in detail.

[](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html)

**java.util.Map** interface extends java.util.Collection interface.

java.util.**[HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*, java.util.***[**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)***, java.util.concurrent.*[ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)*, java.util.*[LinkedHashMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)**, java.util.**[TreeMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)**, java.util.concurrent.**[ConcurrentSkipListMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)**, java.util.**[IdentityHashMap](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html)**, java.util.**[WeakHashMap](http://www.javamadesoeasy.com/2015/04/weakhashmap-in-java.html)**, java.util.**[EnumMap](http://www.javamadesoeasy.com/2015/04/enummap-in-java-with-program.html)** classes implements java.util.**Map** interface.

Also some abstract classes like java.util.**Dictionary** and java.util.**AbstractMap** have been mentioned in hierarchy.

**Collection interview Question 12.  What are differences between** [**Iterator and Enumeration**](http://www.javamadesoeasy.com/2015/04/iterator-vs-enumeration-differences-and.html) **in java?**

**Answer**. Experienced developers must be well versed to answer this collection framework interview question in java.

***Differences*** *between java.util.****Iterator*** *and java.util.****Enumeration*** *in java>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | ***Property*** | ***java.util.Enumeration*** | ***java.util.Iterator*** |
| 1 | Remove elements during iteration | java.util.Enumeration **doesn’t allows** to remove elements from collection during iteration in java. | java.util.Iterator **allows** to remove elements from collection during iteration by using **remove()** method in java. |
| 2 | Improved naming conventions in Iterator | **nextElement()**  Method Returns the next element of this enumeration if this enumeration object has at least one more element to provide.  **hasMoreElements()**  returns true if enumeration contains more elements. | **nextElement()** has been changed to **next()** in Iterator.  And  **hasMoreElements()** has been changed to **hasNext()** in Iterator. |
| 3 | Introduced in  which java  version | Enumeration was introduced in first version  of java i.e. ​**JDK 1.0** | Iterator was introduced in second version  of java i.e. ​**JDK 2.0**  Iterator was introduced to replace Enumeration in the Java Collections Framework. |
| 4 | Recommendation | **Java docs** recommends iterator over enumeration**.** | **Java docs** recommends iterator over enumeration**.** |
| 5 | Enumeration and Iterator over [**Vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html) | **Enumeration** returned by Vector is [**fail-safe**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html), means any modification made to Vector during iteration using Enumeration don’t throw any exception in java. | **Iterator** returned by Vector are [**fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to ArrayList during iteration will throw ConcurrentModificationException  in java. |

**For more detail read :** [**Iterator vs Enumeration - Differences and similarities in java**](http://www.javamadesoeasy.com/2015/04/iterator-vs-enumeration-differences-and.html)

**Collection interview Question 13. How do we override equals and hashcode method in java, write a code to use Employee as key in HashMap in java? (Important)**

**Answer**.  This is one of the most important collection framework interview question in java. Prepare for this question properly. Freshers must know the concept how to override equals and hashcode method but experienced developers must be able to write the java code to override equals and hashcode neatly. We will override equals() and hashCode() like this -

By overriding equals() and hashCode() method we could use custom object as key in HashMap.

1)  Check whether obj is null or not.

**if(obj==null) //If obj is null, return without comparing obj& Employee class.**

2)  check whether  obj is instance of Employee class or not.

**if(this.getClass()!=obj.getClass()) //identifies whether obj is instance of Employee class or not.**

3) Then, type cast obj into employee instance.

**Employee emp=(Employee)obj;  //type cast obj into employee instance.**

|  |
| --- |
| **@Override**  **Public boolean equals(Object obj){**    **if(obj==null)**  **return false;**    **if(this.getClass()!=obj.getClass())**  **return false;**    **Employee emp=(Employee)obj;**  **return (emp.id==this.id || emp.id.equals(this.id))**  **&& (emp.name==this.name || emp.name.equals(this.name));**  **}**    **@Override**  **Public int hashCode(){**  **int hash=(this.id==null ? 0: this.id.hashCode() ) +**  **(this.name==null ? 0: this.name.hashCode() );**  **return hash;**  **}** |

Let’s say in an organisation there exists a employee with **id=1 and name=’sam’**     and **some data** is stored corresponding to him, but if modifications have to be made in data, **previous data must be overridden**.

[DETAILED DESCRIPTION : Override equals() and hashCode() method](http://www.javamadesoeasy.com/2015/02/override-equals-and-hashcode-method.html).

**Must read :** [**Overriding equals and hashcode method - Top 18 Interview questions in java**](http://www.javamadesoeasy.com/2015/02/overriding-equals-and-hashcode-method.html)

**Collection interview Question 14. What classes should i prefer to use a key in HashMap in java? (Important)**

**Answer**. This collection framework interview question will check your in depth knowledge of Java’s Collection Api’s. we should prefer **String, Integer, Long, Double, Float, Short and any other wrapper class.** Reason behind using them as a key is that they override equals() and hashCode() method, we need not to write any explicit code for overriding equals() and hashCode() method in java.

Let’s use Integer class as key in HashMap(Example) -

|  |
| --- |
| **import** java.util.HashMap;  **import** java.util.Map;  **publicclass** StringInMapExample {  **publicstaticvoid** main(String...a){             //HashMap's key=Integer class  (Integer’s api has already overridden hashCode() and equals() method for us )            Map<Integer, String> hm=**new** HashMap<Integer, String>();            hm.put(1, "data");            hm.put(1, "data OVERRIDDEN");              System.*out*.println(hm.get(1));       }  }  /\*OUTPUT  data OVERRIDDEN  \*/ |

If, we note above program, what we will see is we didn’t override equals() and hashCode() method, but still we were able to store data in HashMap, override data and retrieve data using get method.

>Let’s check in **Integer’s API**, how Integer class has overridden equals() and hashCode() method :

|  |
| --- |
| **publicinthashCode**() {  **return**value;  }  **publicbooleanequals**(Object obj) {  **if** (obj **instanceof** Integer) {  **return**value == ((Integer)obj).intValue();         }  **returnfalse**;  } |

**Collection interview Question 15. What are differences between** [**HashMap and Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) **in java?**

**Answer**. Fresher and Experienced developers must answer this important collection framework interview question in detail in java.

***Differences*** *between java.util.****[HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*** *and java.util.****Hashtable*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashMap*** | ***java.util.Hashtable*** |
| 1 | synchronization | java.util.HashMap is **not synchronized**  (because 2 threads on same HashMap object can access it at same time) in java. | java.util.Hashtable is **synchronized** (because 2 threads on same Hashtable object cannot access it at same time) in java. |
| 2 | Performance | HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable in java. | Hashtable is synchronized, hence its operations are **slower** as compared to HashMap in java.  If we are working not working in multithreading environment jdk recommends us to use HashMap. |
| 3 | Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. many keys can have null value in java. | Hashtable does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException) in java. |
| 4 | Introduced  in which java version | HashMap was introduced in second version of java i.e. **JDK 2.0** | Hashtable was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the[Java Collections Framework](http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html). |
| 5 | Recommendation | In non-multithreading environment it is recommended to use HashMap than using Hashtable in java. | I**n java 5 i.e. JDK 1.5**, it is **recommended** to use [ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/concurrenthashmap-in-java.html) than using Hashtable. |
| 6 | Extends Dictionary (Abstract class, which is obsolete) | HashMap does not extends Dictionary in java. | Hashtable extends Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key) in java. |

For more detail read : [**HashMap and Hashtable - Similarity and Differences in java**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)

**Collection interview Question 16. when to use** [**HashSet vs LinkedHashSet vs TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) **in java?**

**Answer**. Another very important collection framework interview question to differentiate between **following Set implementations** in java.

***Differences*** *between java.util.****[HashSet](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)*** *vs java.util.****LinkedHashSet*** *vs java.util.****TreeSet*** *in java>*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Property | *java.util.HashSet* | *java.util.LinkedHashSet* | *java.util.TreeSet* |
| 1 | Insertion order | java.util.HashSet does not maintains insertion order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **No specific order** | java.util.LinkedHashSet maintains insertion order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **b**  **c**  **a** | java.util.TreeSet is sorted by natural order in java.  Example in java >  **set.add("b");**  **set.add("c");**  **set.add("a");**  Output >  **a**  **b**  **c** |
| 2 | Null elements | HashSet allows to store **one null** in java**.** | LinkedHashSet allows to store **one null** in java. | TreeSet does **not** allows to store **any null** in java.  Any attempt to add null throws runtimeException (NullPointerException). |
| 3 | Data structure internally used for storing data | For storing elements HashSet internally uses HashMap. | For storing elements LinkedHashSet internally uses  LinkedHashMap. | For storing elements TreeSet internally uses TreeMap. |
| 4 | Introduced  in which java version | java.util.HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | java.util.LinkedHashSet was introduced in second version of java (1.4) i.e. **JDK 4.0** | java.util.TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0** |
| 5 | Implements which interface | HashSet implements **java.util.**[**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html)interface. | LinkedHashSet implements **java.util.Set** interface. | TreeSet implements **java.util.Set**  **java.util.SortedSet**  **java.util.NavigableSet** interface. |

For more detail read : [**HashSet vs LinkedHashSet vs TreeSet in java**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)

**Collection interview Question 17. What are differences between** [**HashMap and ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) **in java?**

**Answer**. Take my words java developers won’t be able to get away from this very important collection framework interview question.

***Differences*** *between java.util.****[HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*** *and java.util.concurrent.[ConcurrentHashMap](http://www.javamadesoeasy.com/2015/04/concurrenthashmap-in-java.html) in java >*

|  |  |  |
| --- | --- | --- |
| Property | *java.util.****HashMap*** | *java.util.concurrent.* ***ConcurrentHashMap*** |
| synchronization | HashMap is **not synchronized.** | ConcurrentHashMap is **synchronized**. |
| 2 threads on same Map object can access it at concurrently? | Yes, because HashMap is not synchronized**.** | Yes.  But how despite of being synchronized, 2 threads on same *ConcurrentHashMap* object can access it at same time?  *ConcurrentHashMap* is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently. |
| Performance | We will **synchronize HashMap and then compare its performance with ConcurrentHashMap**.  *We can synchronize hashMap by using Collections’s class* ***synchronizedMap*** *method.*   |  | | --- | | *Map synchronizedMap = Collections.****synchronizedMap****(hashMap);* |   *Now, no 2 threads can access same instance of map concurrently.*  **Hence synchronized HashMap’s performance is slower as compared to ConcurrentHashMap.**  But why we didn’t compared HashMap (unSynchronized) with ConcurrentHashMap?  Because performance of unSynchronized collection is always better than some synchronized collection. As, default (unSynchronized) hashMap didn’t cause any locking. | **ConcurrentHashMap’s performance is faster as compared to HashMap (**because it is divided into segments, as discussed in above point**).**  [*Read this post for performance comparison between HashMap and ConcurrentHashMap.*](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) |
| Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. any key can have null value. | ConcurrentHashMap does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException). |
| iterators | The iterators returned by the iterator() method of HashMap are [***fail-fast***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) *>*  *hashMap.keySet().iterator()*  *hashMap.values().iterator()*  *hashMap.entrySet().iterator()*  all three iterators are ***fail-fast*** | iterators are [***fail-safe***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*  *concurrentHashMap.keySet().iterator()*  *concurrentHashMap.values().iterator()*  *concurrentHashMap.entrySet().iterator()*  all three iterators are ***fail-safe.*** |
| **putIfAbsent** | HashMap does not contain putIfAbsent method.  ***putIfAbsent*** *method is equivalent to writing following code >*   |  | | --- | | **synchronized** (map){  **if** (!*map*.containsKey(key))  **return***map*.put(key, value);  **else**  **return***map*.get(key);  } |   [**Program to create method that provides functionality similar to putIfAbsent method of ConcurrentHashMap and to be used with HashMap**](http://www.javamadesoeasy.com/2015/04/program-to-create-method-that-provides.html) | If map does not contain specified **key**, put specified **key-value** pair in map and return null.  If map already contains specified **key**, return value corresponding to specified **key**.    [**Program to use ConcurrentHashMap’s putIfAbsent method**](http://www.javamadesoeasy.com/2015/04/program-to-use-concurrenthashmaps.html) |
| Introduced  in which java version | HashMap was introduced in **java 2 i.e. JDK 1.2**, | ConcurrentHashMap was introduced in **java 5** i.e. **JDK 1.5**, since its introduction Hashtable has become obsolete, because of concurrency level its performance is better than Hashtable. |
| Implements which interface | HashMap implements **java.util.**[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | ConcurrentHashMap implements  **java.util.Map** and  **java.util.concurrent.ConcurrentMap** |
| Package | HashMap is in **java.util** package | ConcurrentHashMap is in **java.util.concurrent** package. |

For more detail read : [**HashMap and ConcurrentHashMap in java**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)

**Collection interview Question 18. When to use** [**HashMap vs Hashtable vs LinkedHashMap vs TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) **in java?**

**Answer**. Another important collection framework interview question

to differentiate between **following Map implementations** in java.

***Differences*** *between java.util.****[HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*** *vs java.util.****[Hashtable](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html)****vs java.util.****LinkedHashMap*** *vs java.util.****[TreeMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)>***

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Property** | ***HashMap*** | ***Hashtable*** | ***LinkedHashMap*** | ***TreeMap*** |
| 1 | Insertion order | HashMap does not maintains insertion order in java. | Hashtable does not maintains insertion order in java. | LinkedHashMap  maintains insertion order in java. | TreeMap is sorted by natural order of keys in java. |
| 2 | Performance | HashMap is not synchronized, hence its operations are **faster** as compared to Hashtable. | Hashtable is synchronized, hence its operations are **slower** as compared HashMap.  If we are working not working in multithreading environment jdk recommends us to use HashMap. | LinkedHashMap must be used only when we want to maintain insertion order. **Time and space overhead** is there because for maintaining order it internally uses **Doubly Linked list**. | TreeMap must be used only when we want sorting based on natural order. Otherwise sorting operations cost performance. (Comparator is called for sorting purpose) |
| 3 | Null keys and values | HashMap allows to store **one null key** and **many null values** i.e. many keys can have null value in java. | Hashtable does **not allow to store null key or null value**.  Any attempt to store null key or value throws runtimeException (NullPointerException) in java. | LinkedHashMap allows to store **one null key** and **many null values** i.e. any key can have null value in java. | TreeMap does **not allow to store null key but allow many null values**.  Any attempt to store null key throws runtimeException (NullPointerException) in java. |
| 4 | Implements which interface | HashMap implements **java.util.**[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | Hashtable implements **java.util.Map** | LinkedHashMap implements **java.util.Map** | TreeMap implements  **java.util.Map**  **java.util.SortedMap**  **java.util.NavigableMap** |
| 5 | Implementation uses? | HashMap use [**buckets**](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html) | Hashtable use **buckets** | LinkedHashMap uses [**doubly linked lists**](http://www.javamadesoeasy.com/2015/02/linkedhashmap-custom-implementation.html) | TreeMap uses **Red black tree** |
| 6 | Complexity of put, get and remove methods | O(1) | O(1) | O(1)  **overhead** of updating **Doubly Linked list** for maintaining order it internally uses. | O(log(n)) |
| 7 | Extends java.util.**Dictionary** (Abstract class, which is obsolete) | HashMap **doesn’t** extends Dictionary. | Hashtable **extends** Dictionary (which maps non-null keys to values. In a given Dictionary we can look up value corresponding to key) | LinkedHashMap **doesn’t** extends Dictionary. | TreeMap **doesn’t** extends Dictionary. |
| 8 | Introduced in which java version? | HashMap was introduced in second version of java i.e. **JDK 2.0** | Hashtable was introduced in first version of java i.e. **JDK 1.0**  But it was refactored in java 2 i.e. JDK 1.2 to implement the Map interface, hence making it a member of member of the[Java Collections Framework](http://download.oracle.com/javase/7/docs/technotes/guides/collections/index.html). | LinkedHashMap was introduced in fourth version of java i.e. **JDK 4.0** | TreeMap was introduced in second version of java i.e. **JDK 2.0** |

For more detail read : [**HashMap vs Hashtable vs LinkedHashMap vs TreeMap in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)

**Collection interview Question 19. What are differences between** [**HashMap vs IdentityHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html) **in java?**

**Answer**. This is tricky and complex collection framework interview question for experienced developers in java.

***Differences*** *between java.util.****[HashMap](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)****and java.util.****[IdentityHashMap](http://www.javamadesoeasy.com/2015/04/identityhashmap-in-java.html)*** *in java>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashMap*** | ***java.util.IdentityHashMap*** |
| 1 | **Keys comparison***object-equality  vs reference-equality* | **HashMap** when comparing keys (and values) performs object-equality not reference-equality. In an HashMap, two keys k1 and k2 are equal if and only if (k1==null ? k2==null : k1.equals(k2)) | **IdentityHashMap** when comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if (k1==k2) |
| 2 | Initial size | Constructs a new HashMap, Its initial capacity is 16 in java.   |  | | --- | | **new** HashMap(); | | Constructs a new IdentityHashMap, with maximum size of 21 in java.   |  | | --- | | **new** IdentityHashMap(); | |
| 3 | Introduced in which java version | HashMap was introduced in second version of java i.e. **JDK 2.0** | IdentityHashMap was introduced in fourth version of java i.e. **JDK 4.0** |
| 4 | *Program* | Program 1 shows > *comparing keys (and values) performs object-equality in place of reference-equality . In an HashMap, two keys k1 and k2 are equal if and only if* **(k1==null ? k2==null : k1.equals(k2)).** | Program 2 shows >  *comparing keys (and values) performs reference-equality in place of object-equality. In an IdentityHashMap, two keys k1 and k2 are equal if and only if* **(k1==k2).** |
| 5 | overridden equals() and hashCode() method call? | [*overridden equals() and hashCode() method*](http://www.javamadesoeasy.com/2015/02/override-equals-and-hashcode-method.html)are called when put, get methods are called in ***HashMap***.  As shown in Program 3. | *overridden equals() and hashCode() method* are not called when put, get methods are called in ***IdentityHashMap***.  *Because IdentityHashMap implements equals() and hashCode() method by itself and checks for reference-equality of keys.*  As shown in Program 4. |
| 6 | Application - can maintain *proxy object* | HashMap cannot be used to maintain *proxy object.* | IdentityHashMap can be used to maintain *proxy objects*. For example, we might need to maintain proxy object for each object debugged in the program. |

For more detail read : [**HashMap vs IdentityHashMap - Similarity and Differences with program in java**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-identityhashmap-similarity.html)

**Collection interview Question 20. What is WeakHashMap in java?**

**Answer**.  Another tricky collection framework interview question for experienced developers in java.

java.util.[WeakHashMap](http://www.javamadesoeasy.com/2015/04/weakhashmap-in-java.html) is hash table based implementation of the Map interface, with *weak keys*.

An entry in a WeakHashMap will be automatically removed by garbage collector when its key is no longer in ordinary use. Mapping for a given key will not prevent the key from being discarded by the garbage collector, (i.e. made finalizable, finalized, and then reclaimed). When a key has been discarded its entry is removed from the map in java.

java.util.**WeakHashMap** is implementation of the java.util.[**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) interface in java.

*The behavior of the java.util.WeakHashMap class depends upon garbage collector*

The behavior of the WeakHashMap class depends upon garbage collector in java. Because the garbage collector may discard keys at any time, in WeakHashMap it may look like some unknown thread is silently removing entries. Even if you synchronize WeakHashMap instance and invoke none of its methods,

* it is possible for the **size** method to return smaller values over time,
* for **isEmpty** method to return false and then true,
* for **containsKey** method to return true and later false for a given key,
* for **get** method to return a value for a given key but later return null,
* for **put** method to return null, and
* for **remove** method to return false for a key that previously existed in the WeakHashMap.

Each key object in a WeakHashMap is stored indirectly as the referent of a weak reference. Therefore a key will be removed automatically only after the weak references to it, both inside and outside of the map, have been cleared by the garbage collector.

**Collection interview Question 21. What is EnumSet in java?**

**Answer**. Freshers must know about EnumMap in java.

A java.util.[EnumSet](http://www.javamadesoeasy.com/2015/04/enumset-in-java-with-program.html) is specialized [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) implementation for use with enum types in java.

EnumSet all elements comes from a single enum type that is specified when the set is created in java.

*Order of elements in EnumSet in java*

The java.util.EnumSet maintains ***natural order***(the order in which the enum constants are declared) of elements in java.

*Iterator on EnumSet in java*

The iterator returned by the iterator method traverses the elements in their ***natural order***(the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the set that occur while the iteration is in progress.

*Null elements in EnumSet in java*

Null elements are not allowed in EnumSet in java. Attempts to insert a null element will throw NullPointerException in java.

**Collection interview Question 22. What is EnumMap in java?**

**Answer**. Freshers must be able to answer this collection framework interview question in java. A java.util.[EnumMap](http://www.javamadesoeasy.com/2015/04/enummap-in-java-with-program.html) is specialized [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) implementation for use with enum type keys.

EnumMap all keys comes from a single enum type that is specified when the set is created in java.

*Order of keys in EnumMap in java*

The EnumMap maintains ***natural order***(the order in which the enum constants are declared) of keys in java.

*Iterator on EnumMap in java*

The iterator returned by the iterator method in EnumMap traverses the elements in their ***natural order* of keys**(the order in which the enum constants are declared).

iterator never throw ConcurrentModificationException and it may or may not show the effects of any modifications to the map that occur while the iteration is in progress in java.

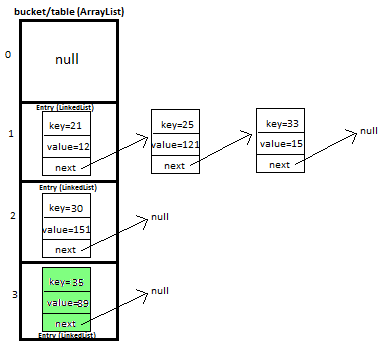
*Null allowed in EnumMap in java?*

**Null keys are not allowed** in EnumMap. Attempts to insert a null key will throw NullPointerException.

**But, Null values are allowed** in EnumMap in java.

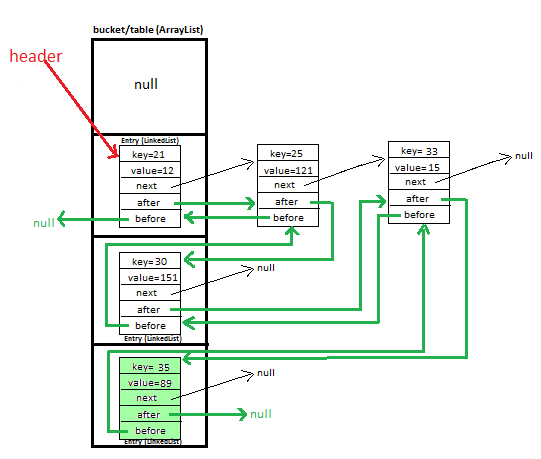
**How to implement own/custom HashMap in java? Or How HashMap works in java?**

**Answer**.

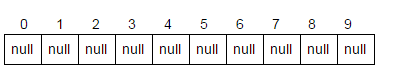
[HashMap Custom implementation/ HashMap works in java](http://javamadesoeasy.com/2015/02/hashmap-custom-implementation.html)

**Collection interview Question 24. How to implement own LinkedHashMap in java? Or LinkedHashMap works in java?**

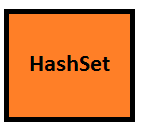
**Answer**.

[LinkedHashMap Custom implementation/How LinkedHashMap works in java](http://javamadesoeasy.com/2015/02/linkedhashmap-custom-implementation.html)

**Collection interview Question 25. How to implement own ArrayList in java?Or How ArrayList works in java ?**

**Answer**. [ArrayList custom implementation /  How ArrayList works in java](http://javamadesoeasy.com/2015/02/arraylist-custom-implementation.html)

**Collection interview Question 26. How to implement own HashSet in java? Or How HashSet works in java ?**

**Answer**. [****Set Custom implementation/ Or How HashSet works in java](http://javamadesoeasy.com/2015/02/set-custom-implementation.html)

**Collection interview Question 27. How to implement own LinkedHashSet in java? Or How LinkedHashSet works in java ?**

**Answer**. [LinkedHashSet Custom implementation/ How LinkedHashSet works in java](http://javamadesoeasy.com/2015/02/linkedhashset-custom-implementation.html)

**Collection interview Question 28. What do you mean by fail-fast and fast-safe? What is ConcurrentModificationException?**

**Answer**.

Iterator returned by few Collection framework Classesare **fail-fast,** means any structural modification made to these classes during iteration will throw ConcurrentModificationException.

Some important classes whose returned iterator is **fail-fast >**

* [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)
* [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html)
* [**vector**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-vector-similarity-and.html)
* [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)

Iterator returned by few Collection framework Classes are **fail-safe,** means any structural modification made to these classes during iteration won’t throw any Exception.

Some important classes whose returned iterator is **fail-safe >**

* [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)
* [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)
* [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)

**For more detail read :** [**ConcurrentModificationException, Fail-fast and Fail-safe in detail in java**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)

**Collection interview Question 29. What are different ways of iterating over elements in List?**

**Answer**.

*Creating* [*ArrayList*](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) *and add element.*

|  |
| --- |
| List<String> arrayList=**new** ArrayList<String>();  arrayList.add("javaMadeSoEasy"); |

1. *Iterate over elements in ArrayList using* ***iterator****()*

iterator() method returns iterator to iterate over elements in ArrayList.

|  |
| --- |
| Iterator<String> iterator=arrayList.iterator();  **while**(iterator.hasNext()){            System.*out*.println(iterator.next());     } |

*iterator returned by ArrayList is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*

1. *Iterate over elements in ArrayList using* ***listIterator****()*

|  |
| --- |
| ListIterator<String> listIterator=arrayList.listIterator(); |

***ListIterator returned by ArrayList is also fail fast***.

1. *Iterate over elements in list using* ***enumeration***

|  |
| --- |
| Enumeration<String> listEnum=Collections.*enumeration*(arrayList);  **while**(listEnum.hasMoreElements()){        System.*out*.println(listEnum.nextElement());     } |

enumeration is also fail-fast.

1. *Iterate over elements in list using* ***enhanced for loop***

|  |
| --- |
| **for** (String string : arrayList) {               System.*out*.println(string);        } |

**enhanced for loop** is also fail-fast.

**Collection interview Question 30. What are different ways of iterating over elements in Set?**

**Answer**. *Creating* [*HashSet*](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) *and add element.*

|  |
| --- |
| Set<String> hashSet=**new** HashSet<String>();  hashSet.add("javaMadeSoEasy"); |

1. *Iterate over elements in HashSet using iterator()*

iterator() method returns iterator to iterate over elements in HashSet.

|  |
| --- |
| Iterator<String> iterator=hashSet.iterator();  **while**(iterator.hasNext()){            System.*out*.println(iterator.next());     } |

*iterator returned by HashSet is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*.*

1. *Iterate over elements in Set using enumeration*

|  |
| --- |
| Enumeration<String> listEnum=Collections.*enumeration*(set);  **while**(listEnum.hasMoreElements()){        System.*out*.println(listEnum.nextElement());     } |

enumeration is also fail-fast.

1. *Iterate over elements in Set using* ***enhanced for loop***

|  |
| --- |
| **for** (String string : set) {               System.*out*.println(string);        } |

**enhanced for loop** is also fail-fast.

**Collection interview Question 31. What are different ways of iterating over keys, values and entry in Map?**

**Answer**. *Create and put key-value pairs in* [*HashMap*](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html)*>*

|  |
| --- |
| Map<Integer,String> hashMap=**new** HashMap<Integer,String>();  hashMap.put(11, "javaMadeSoEasy");  hashMap.put(21, "bmw");  hashMap.put(31, "ferrari"); |

1. Iterate over keys -

***hashMap.keySet().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<Integer> keyIterator=hashMap.keySet().iterator();  **while**(keyIterator.hasNext()){   System.*out*.println(keyIterator.next());  }  /\*OUTPUT  21  11  31  \*/ |

1. Iterate over values -

***hashMap.values().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<String> valueIterator=hashMap.values().iterator();  **while**(valueIterator.hasNext()){   System.*out*.println(valueIterator.next());  }  /\*OUTPUT  javaMadeSoEasy  audi  ferrari  \*/ |

*iterator returned is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*..*

1. Iterate over Entry-

***hashMap.entrySet().iterator()*** method returns iterator to iterate over keys in HashMap.

|  |
| --- |
| Iterator<Entry<Integer, String>> entryIterator=hashMap.entrySet().iterator();  **while**(entryIterator.hasNext()){     System.*out*.println(entryIterator.next());  }  /\*OUTPUT  21=javaMadeSoEasy  11=audi  31=ferrari  \*/ |

*iterator returned is* [*fail-fast*](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*..*

**32. What is difference between** [**Comparable and Comparator**](http://www.javamadesoeasy.com/2015/04/comparable-vs-comparator-differences.html)**? How can you sort List?**

**Answer**.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***Comparable*** | ***Comparator*** |
| 1 | Comparing instances of class | Comparable is used to compare instances of same class | Comparator can be used to compare instances of same or different classes. |
| **2** | **sorting order** | Comparable can be implemented by class which need to define a **natural ordering for its objects.**  **Example** - String, Integer, Long , [Date](http://www.javamadesoeasy.com/2015/07/creating-date-in-java-using-calendar.html) and all other wrapper classes implements Comparable. | Comparator is implemented when one wants a **different sorting order** and define custom way of comparing two instances. |
| 3 | Changes to class | For using Comparable, original Class must implement it.    **Example-**  **class** Employee **implements Comparable<Employee>**    For using Comparable, Employee Class must implement it, no other class can implement it.  As used in **Program 1** | Class itself can implement Comparator  or  any other class can implement Comparator. Hence avoiding modification to original class.  **Example-**  **class ComparatorName implements Comparator<Employee>**  **class ComparatorId implements Comparator<Employee>**  In above example modifications were made to **ComparatorName** and **ComparatorId.** Hence avoiding modification to Employee class.  As used in **Program 4** |
| 4 | Sorting on basis on one or many criteria | Provides sorting only on **one** criteria, **because** Comparable can be implemented by original class only. | We can use Comparator to sort class on **many** criterias **because** class itself or any other class can implement Comparator. |
| 5 | Method | compareTo method  **@Override**  **publicint compareTo(Employee obj) {**  **//sort Employee on basis of name(ascending order)**  **returnthis.name.compareTo(obj.name);**  **}**  Method compares **this** with **obj** object and returns a integer.   * positive – **this** is **greater** than **obj** * zero – **this** is **equal** to **obj** * negative – **this** is **less** than **obj**     As used in **Program 1** | compare method  **@Override**  **publicint compare(Employee obj1, Employee obj2) {**  **//sort Employee on basis of name(ascending order)**  **return obj1.name.compareTo(obj2.name);**  **}**    Method compares **obj1** with **obj2** object and returns a integer.   * positive – **obj1** is **greater** than **obj2** * zero – **obj1** is **equal** to **obj2** * negative – **obj1** is **less** than **obj2**     As used in **Program 3** |
| 6 | Package | **java.lang**  **java.lang** package is automatically imported by every program in java.  Hence, we need to write explicit statement for importing java.lang.Comparable. | **java.util**  We need to write explicit import statement -  **import** java.util.Comparator |
| 7 | Using **Collections.sort** | Let's say we wanna sort list of Employee,  **Collections.sort(**list**)** uses Comparable interface for sorting class.  As used in Program 1 | Let's say we wanna sort list of Employee,  **Collections.*sort*(list,new ComparatorName());**  uses Comparator interface for sorting class.  As used in Program 5 |

Read more : [**Comparable vs Comparator - differences and sorting list by implementing Comparable and Comparator in classes and inner classes**](http://www.javamadesoeasy.com/2015/04/comparable-vs-comparator-differences.html)

**. How sort method of Collections class works internally?**

**Answer**. *Collections.sort internally calls Arrays.sort,*

*Arrays.Sort() internally uses* [*Merge Sort*](http://javamadesoeasy.blogspot.in/2015/01/merge-sort.html)*.*

If number of elements is less than 7 then [Insertion Sort](http://www.javamadesoeasy.com/2015/01/insertion-sort.html) is used rather than [*Merge Sort*](http://javamadesoeasy.blogspot.in/2015/01/merge-sort.html). (because in case elements are less than 7 it offers better time complexity)

**Collection interview Question 34. How can you sort given HashMap on basis of keys?**

**Answer**.

**Please Read :** [**Sort Map by key in Ascending and descending order by implementing Comparator interface and overriding its compare method and using TreeMap**](http://www.javamadesoeasy.com/2015/04/sort-map-by-key-in-ascending-and.html)

**Collection interview Question 35. How can you sort given HashMap on basis of values?**

**Answer**.

**Please Read :** [**Sort Map by value in Ascending and descending order by implementing Comparator interface and overriding its compare method**](http://www.javamadesoeasy.com/2015/04/sort-map-by-value-in-ascending-and.html)

**Collection interview Question 36. In what all possible ways you can sort a given Set?**

**Answer**.

**Please Read :** [**Sort Set by using TreeSet and by implementing Comparator and Comparable interface**](http://www.javamadesoeasy.com/2015/04/sort-set-by-using-treeset-and-by.html)

**Collection interview Question 37. How you can sort arrays? And how Comparator of superclass can be used by subclasses?**

**Answer**.

**Please Read :** [**Arrays.sort to sort arrays by implementing Comparator and how Comparator of superclass can be used by subclasses**](http://www.javamadesoeasy.com/2015/04/arrayssort-to-sort-arrays-by.html)

**Collection interview Question 38. What are differences between** [**ArrayList vs CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)**?**

**Answer**.

***Differences*** *between java.util.****[ArrayList](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html)****and java.util.concurrent.****CopyOnWriteArrayList****in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.ArrayList*** | **java.util.concurrent. CopyOnWriteArrayList** |
| 1 | synchronization | ArrayList is not **synchronized**  (because 2 threads on same ArrayList object can access it at same time).  I have created **program** to show see consequence of using ArrayList in multithreading environment.  In the program i will implement our own arrayList. | **CopyOnWriteArrayList**is **synchronized**  (because 2 threads on same CopyOnWriteArrayList object cannot access it at same time). |
| 2 | Iterator and listIterator | Iterator and listIterator returned by ArrayList are [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to ArrayList during iteration using Iterator or listIterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator and listIterator returned by CopyOnWriteArrayList are **Fail-safe** in java.  As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by ArrayList is **fail-fast**, means any structural modification made to ArrayList during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by CopyOnWriteArrayList is **fail-safe.**    As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on ArrayList using **enhanced for loop** is **Fail-fast,** means any structural modification made to ArrayList during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on CopyOnWriteArrayList using **enhanced for loop** is **Fail-safe.**  As shown in Program 2 below. |
| 5 | Performance | ArrayList is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArrayList. | CopyOnWriteArrayList is synchronized, hence its operations are **slower** as compared to ArrayList. |
| 6 | AbstractList | ArrayList extends AbstractList (abstract class) which provides implementation to  List interface to minimize the effort required to implement this interface backed by RandomAccess interface. | CopyOnWriteArrayList does not extends AbstractList, though CopyOnWriteArrayList also implements RandomAccess interface. |
| 7 | Introduced in which java version | ArrayList was introduced in second version of java (1.2) i.e. **JDK 2.0** | CopyOnWriteArrayList was introduced in fifth version of java (1.5) i.e. **JDK 5.0** |
| 8 | Package | java.util | java.util.**concurrent** |

**For more detail read :**[ArrayList vs CopyOnWriteArrayList - Similarity and Differences with program](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)

**Collection interview Question 39. What are differences between** [**HashSet vs CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)**?**

**Answer**.

***Differences*** *between java.util.****[HashSet](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html)****and java.util.concurrent.****CopyOnWriteArraySet*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.HashSet*** | **java.util.concurrent. CopyOnWriteArraySet** |
| 1 | synchronization | HashSet is not **synchronized**  (because 2 threads on same HashSet object can access it at same time) in java. | **CopyOnWriteArraySet**is **synchronized**  (because 2 threads on same CopyOnWriteArraySet object cannot access it at same time) in java. |
| 2 | Iterator | Iterator returned by HashSet is [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to HashSet during iteration using Iterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator returned by **CopyOnWriteArraySet** is **Fail-safe** in java.  As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by HashSet is **fail-fast**, means any structural modification made to HashSet during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by CopyOnWriteArraySet is **fail-safe.**  As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on HashSet using **enhanced for loop** is **Fail-fast,** means any structural modification made to HashSet during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on CopyOnWriteArraySet using **enhanced for loop** is **Fail-safe.**    As shown in Program 2 below. |
| 5 | Performance | HashSet is not synchronized, hence its operations are **faster** as compared to CopyOnWriteArraySet. | CopyOnWriteArraySet is synchronized, hence its operations are **slower** as compared to HashSet. |
| 6 | Introduced in which java version | HashSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | CopyOnWriteArraySet  was introduced in fifth version of java (1.5) i.e. **JDK 5.0** |
| 7 | Package | java.util | java.util.**concurrent** |

***For more detail read :*** [**HashSet vs CopyOnWriteArraySet - Similarity and Differences with program**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)

**Collection interview Question 40. What are differences between** [**TreeSet vs ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)**?**

**Answer**.

***Differences*** *between java.util.****[TreeSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)*** *and java.util.concurrent.****ConcurrentSkipListSet*** *in java>*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.TreeSet*** | **java.util.concurrent. ConcurrentSkipListSet** |
| 1 | synchronization | TreeSet is not **synchronized**  (because 2 threads on same TreeSet object can access it at same time) in java. | **ConcurrentSkipListSet**is **synchronized**  (because 2 threads on same ConcurrentSkipListSet object cannot access it at same time) in java. |
| 2 | Iterator | Iterator returned by TreeSet is [**Fail-fast**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)**,** means any structural modification made to TreeSet during iteration using Iterator will throw ConcurrentModificationException in java.  As shown in Program 1 below. | Iterator returned by **ConcurrentSkipListSet** is **Fail-safe** in java.    As shown in Program 2 below. |
| 3 | Enumeration is fail-fast | **Enumeration** returned by TreeSet is **fail-fast**, means any structural modification made to TreeSet during iteration using Enumeration will throw ConcurrentModificationException.  As shown in Program 1 below. | **Enumeration** returned by ConcurrentSkipListSet is **fail-safe.**    As shown in Program 2 below. |
| 4 | Iterate using **enhanced for loop** | Iteration done on TreeSet using **enhanced for loop** is **Fail-fast,** means any structural modification made to TreeSet during iteration using **enhanced for loop** will throw ConcurrentModificationException.  As shown in Program 1 below. | Iteration done on ConcurrentSkipListSet using **enhanced for loop** is **Fail-safe.**  As shown in Program 2 below. |
| 5 | Performance | TreeSet is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListSet. | ConcurrentSkipListSet is synchronized, hence its operations are **slower** as compared to TreeSet. |
| 6 | Introduced in which java version | TreeSet was introduced in second version of java (1.2) i.e. **JDK 2.0** | ConcurrentSkipListSet was introduced in sixth version of java (1.6) i.e. **JDK 6.0** |
| 7 | Package | java.util | java.util.**concurrent** |

***For more detail read :*** [**TreeSet vs ConcurrentSkipListSet - Similarity and Differences with program**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)

**Collection interview Question 41. What are differences between** [**TreeMap vs ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)**?**

**Answer**.

***Differences*** *between java.util.****[TreeMap](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html)****and java.util.concurrent.****ConcurrentSkipListMap*** *in java >*

|  |  |  |  |
| --- | --- | --- | --- |
|  | Property | ***java.util.TreeMap*** | ***java.util.concurrent. ConcurrentSkipListMap*** |
| 1 | synchronization | TreeMap is **not synchronized**  (because 2 threads on same TreeMap object can access it at same time) in java. | ConcurrentSkipListMap is **synchronized** (because 2 threads on same ConcurrentSkipListMap object cannot access it at same time) in java. |
| 2 | Iterator | The iterators returned by the iterator() method of Map's “collection view methods" are [***fail-fast***](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html)*>*   * *map.keySet().iterator()* * *map.values().iterator()* * *map.entrySet().iterator()*   all three iterators are ***fail-fast*,** means any structural modification made to TreeMap during iteration using any of 3 Iterator will throw ConcurrentModificationException.  As shown in Program 1 below. | The iterators returned by the iterator() method of Map's “collection view methods" are *fail-safe >*   * *map.keySet().iterator()* * *map.values().iterator()* * *map.entrySet().iterator()*   all three iterators are ***fail-safe.***    As shown in Program 2 below. |
| 3 | Performance | TreeMap is not synchronized, hence its operations are **faster** as compared to ConcurrentSkipListMap. | ConcurrentSkipListMap is synchronized, hence its operations are **slower** as compared to TreeMap. |
| 4 | Introduced inin which java version | TreeMap was introduced in second version of java i.e. **JDK 2.0** | ConcurrentSkipListMap was introduced in sixth version of java i.e. **JDK 6.0** |
| 5 | Package | java.util | java.util.**concurrent** |
| 6 | Implements which interface | Map  SortedMap  NavigableMap | Map  SortedMap  NavigableMap  ConcurrentNavigableMap |

***For more detail read :*** [**TreeMap vs ConcurrentSkipListMap - Similarity and Differences with program**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)

**Collection interview Question 43. Can we use null element in TreeSet? Give reason?**

**Answer**. No, [TreeSet](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) does **not** allows to store **any null keys**.

Any attempt to add null throws runtimeException (NullPointerException).

TreeSet internally compares elements for sorting elements by natural order ([comparator may be used for sorting](http://www.javamadesoeasy.com/2015/04/program-to-sort-set-in-ascending-order_24.html), if defined at creation time)and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

**Collection interview Question 44. Can we use null key in TreeMap? Give reason?**

**Answer**. No, [TreeMap](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)**not allow to store null key.**

Any attempt to store null key throws runtimeException (NullPointerException).

TreeMap internally compares keys for sorting keys by natural order ([comparator may be used for sorting](http://www.javamadesoeasy.com/2015/04/program-to-sort-set-in-ascending-order_24.html), if defined at creation time) and null is not comparable, Any attempt to compare null with other object will throw NullPointerException.

**Collection interview Question 45.  How ConcurrentHashMap works? Can 2 threads on same ConcurrentHashMap object access it concurrently?**

**Answer**. [*ConcurrentHashMap*](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)is divided into different **segments** based on concurrency level. So different threads can access different **segments** concurrently.

**Can threads read the segment locked by some other thread?**

Yes. When thread locks one segment for updation it does not block it for retrieval (done by get method) hence some other thread can read the segment (by get method), but it will be able to read the data before locking.

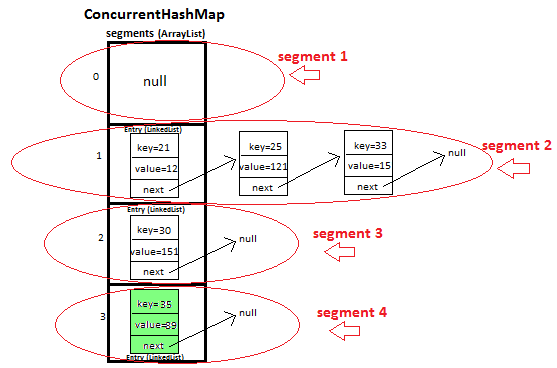
For operations such as putAll concurrent retrievals may reflect removal of only some entries.

For operations such as clear concurrent retrievals may reflect removal of only some entries.

***Segments*** *in ConcurrentHashMap with* ***diagram*** *>*

we have ConcurrentHashMap with **4 segments -**

(Diagram shows how **segments** are formed in ConcurrentHashMap)

**[](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)**

**Collection interview Question 46. Write a program to show consequence of using ArrayList in multithreading environment?**

**Answer.** Program to show [consequence of using ArrayList in multithreading environment in java](http://www.javamadesoeasy.com/2015/05/consequence-of-using-arraylist-in.html)

[dri](https://docs.google.com/document/d/10tUFPo2Xcw5JiwNR_bqIGEw52V6eUGP0KvE2nTHkb-U/edit) [blog](https://www.blogger.com/blogger.g?blogID=5056459490283781613#editor/target=post;postID=4105800325043770563;onPublishedMenu=allposts;onClosedMenu=allposts;postNum=0;src=link)

**Collection interview Question 47. Write a program to show advantage of using Vector in multithreading environment?**

**Answer.**  Program to show [advantage of using Vector in multithreading environment in java](http://www.javamadesoeasy.com/2015/05/advantage-of-using-vector-in.html)

**Collection interview Question 48. Mention properties of most frequently used Collection classes and Interfaces? Mention as many properties as much you can.**

**Answer**. This question is real test for experienced developers, this will test your in depth awareness of Collection classes and Interfaces. Answering this question in detail will really ensure your selection.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| [**List**](http://www.javamadesoeasy.com/2015/04/list-hierarchy-in-java-detailed.html) | **Duplicate elements** | **insertion order** | **Sorted by natural order** | **synchronized** | **null elements** | **Iterator** |
| [**ArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-in-java.html) | Yes | Yes |  |  | Yes | Iterator & listIterator  are  [Fail-fast](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) |
| [**LinkedList**](http://www.javamadesoeasy.com/2015/04/linkedlist-in-java.html) | Yes | Yes |  |  | Yes | Iterator & listIterator  are  Fail-fast |
| [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html) | Yes | Yes |  | Yes | Yes | Iterator & listIterator  are  [**Fail-safe**](http://www.javamadesoeasy.com/2015/04/concurrentmodificationexception-fail.html) |
|  |  |  |  |  |  |  |
| [**Set**](http://www.javamadesoeasy.com/2015/04/set-hierarchy-in-java-detailed-hashset.html) | **Duplicate elements** | **insertion order** | **Sorted by natural order** | **synchronized** | **null elements** | **Iterator** |
| [**HashSet**](http://www.javamadesoeasy.com/2015/04/hashset-in-java.html) |  |  |  |  | Yes | Fail-fast |
| [**LinkedHashSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  | Yes |  |  | Yes | Fail-fast |
| [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) |  |  | Yes |  | No | Fail-fast |
| [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html) |  |  | Yes | Yes | No | **Fail-safe** |
|  |  |  |  |  |  |  |
| [**Map**](http://www.javamadesoeasy.com/2015/04/map-hierarchy-in-java-detailed-hashmap.html) | **Duplicate Keys** | **insertion order of keys** | **Sorted by natural order of keys** | **synchronized** | **null keys or null values** | **Iterator**  **Map implementations returns 3 iterators >**  *map.keySet().iterator()*  *map.values().iterator()*  *map.entrySet().iterator()* |
| [**HashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-in-java.html) |  |  |  |  | one null key and many null values | All are Fail-fast |
| [**Hashtable**](http://www.javamadesoeasy.com/2015/04/hashmap-and-hashtable-similarity-and.html) |  |  |  | Yes | No | All are Fail-fast |
| [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html) |  |  |  | Yes | No | All are **Fail-safe** |
| [**TreeMap**](http://www.javamadesoeasy.com/2015/04/hashmap-vs-hashtable-vs-linkedhashmap.html) |  |  | Yes |  | Null key not allowed,  Allow many null values | All are Fail-fast |
| [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) |  |  | Yes | Yes | No | All are **Fail-safe** |

[**Collection - List, Set and Map all properties in tabular form**](http://www.javamadesoeasy.com/2015/04/collection-list-set-and-map-all.html)

**Collection interview Question. 49 Which list class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**CopyOnWriteArrayList**](http://www.javamadesoeasy.com/2015/04/arraylist-vs-copyonwritearraylist.html)

**Collection interview Question 50. Which Set class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**CopyOnWriteArraySet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-copyonwritearrayset.html)(allows null and elements aren't sorted in natural order) **or** [**ConcurrentSkipListSet**](http://www.javamadesoeasy.com/2015/04/treeset-vs-concurrentskiplistset.html)(doesn’t allows null and elements are sorted in natural order)

**Select one depending on your requirement.**

**Collection interview Question 51. Which Map class must be preferred in multithreading environment, considering performance constraint?**

**Answer**. [**ConcurrentHashMap**](http://www.javamadesoeasy.com/2015/04/hashmap-and-concurrenthashmap.html)(keys aren't sorted in natural order) **or** [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html)(keys are sorted in natural order)

**Select one depending on your requirement.**

**Collection interview Question 52. Let’s say you have to build dictionary and multiple users can add data in that dictionary? And you can use 2 Collection classes? Which Collection classes you will prefer and WHY?**

**Answer**. It’s very **important question** which test your **logical** reasoning and your ability to create robust applications in [multithreading](http://www.javamadesoeasy.com/2015/03/what-is-thread-in-java.html) environment.

We must use [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) and [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html)  >

|  |
| --- |
| **ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =**  **new ConcurrentSkipListMap<String, TreeSet<String>>();** |

Store words in [**ConcurrentSkipListMap**](http://www.javamadesoeasy.com/2015/04/treemap-vs-concurrentskiplistmap.html) as key>

* keys are sorted in **natural order** (words will be sorted in natural order),
* **doesn’t allow null** keys (words can’t be null)
* **doesn’t allow duplicate** keys (words can’t be duplicate) and
* [synchronized](http://www.javamadesoeasy.com/2015/03/synchronization-blocks-and-methods.html), so 2 threads won’t create synchronization problems (will take care of different uses adding words concurrently)

for storing meaning of word in dictionary we must use [**TreeSet**](http://www.javamadesoeasy.com/2015/04/hashset-vs-linkedhashset-vs-treeset.html) as value in ConcurrentSkipListMap **because one word can have many meanings**>

* elements are sorted in **natural order** (meaning of word are sorted in natural order),
* **doesn’t allow null** elements (meaning of word can’t be null),
* **doesn’t allow duplicate** elements (meaning of word can’t be duplicate)

**Program for creating and using Java dictionary using Collection classes>**

|  |
| --- |
| **package** com.ankit.dictionary;  **import** java.util.TreeSet;  **import** java.util.concurrent.ConcurrentSkipListMap;  /\*\* Copyright (c), AnkitMittal JavaMadeSoEasy.com \*/  **publicclass** MyDictionary {  **publicstaticvoid** main(String[] args) {  **ConcurrentSkipListMap<String, TreeSet<String>> myDictionary =**  **new ConcurrentSkipListMap<String, TreeSet<String>>();**            TreeSet<String> innocentMeaning = **new** TreeSet<String>();            innocentMeaning.add("not responsible for an event yet suffering its consequences");            innocentMeaning.add("not guilty of a crime");            myDictionary.put("innocent", innocentMeaning);            TreeSet<String> appealingMeaning = **new** TreeSet<String>();            appealingMeaning.add("attractive");            appealingMeaning.add("expressing a desire for help");            myDictionary.put("appealing", appealingMeaning);            System.*out*.println(myDictionary);     }  }  /\* OUTPUT  {**appealing**=[attractive, expressing a desire for help], **innocent**=[not guilty of a crime, not responsible for an event yet suffering its consequences]}  \*/ |

**Collection interview Question 53. Why to use java.util.WeakHashMap map which is so inconsistent and unpredictable in behaviour?**

**Answer**. Let's say we have huge application which consists of lots n lots of object and may run short of memory at any time, we will like [garbage collector](http://www.javamadesoeasy.com/2015/09/how-garbage-collection-works-internally.html) to quickly discard less used key value pair to free up some memory. As, behavior of the WeakHashMap class depends upon garbage collector.

I believe discarding less used key-value is always going to a better option than running out of memory.