Java Basic question

|  |
| --- |
| ClassLoader in Java  Java ClassLoader  Java ClassLoader is an abstract class. It belongs to a **java.lang** package. It loads classes from different resources. Java ClassLoader is used to load the classes at run time. In other words, JVM performs the linking process at runtime. Classes are loaded into the JVM according to need. If a loaded class depends on another class, that class is loaded as well. When we request to load a class, it delegates the class to its parent. In this way, uniqueness is maintained in the runtime environment. It is essential to execute a Java program.  ClassLoader in Java  Java ClassLoader is based on three principles: **Delegation**, **Visibility**, and **Uniqueness**.   * **Delegation principle:** It forwards the request for class loading to parent class loader. It only loads the class if the parent does not find or load the class. * **Visibility principle:** It allows child class loader to see all the classes loaded by parent ClassLoader. But the parent class loader cannot see classes loaded by the child class loader. * **Uniqueness principle:** It allows to load a class once. It is achieved by delegation principle. It ensures that child ClassLoader doesn't reload the class, which is already loaded by the parent.   Types of ClassLoader  In Java, every ClassLoader has a predefined location from where they load class files. There are following types of ClassLoader in Java:  **Bootstrap Class Loader:** It loads standard JDK class files from rt.jar and other core classes. It is a parent of all class loaders. It doesn't have any parent. When we call String.class.getClassLoader() it returns null, and any code based on it throws NullPointerException. It is also called Primordial ClassLoader. It loads class files from jre/lib/rt.jar. For example, java.lang package class.  **Extensions Class Loader:** It delegates class loading request to its parent. If the loading of a class is unsuccessful, it loads classes from jre/lib/ext directory or any other directory as java.ext.dirs. It is implemented by sun.misc.Launcher$ExtClassLoader in JVM.  **System Class Loader:** It loads application specific classes from the CLASSPATH environment variable. It can be set while invoking program using -cp or classpath command line options. It is a child of Extension ClassLoader. It is implemented by sun.misc.Launcher$AppClassLoader class. All Java ClassLoader implements java.lang.ClassLoader.  5.6M  93  Hello Java Program for Beginners  **Next**  **Stay**  ClassLoader in Java  How ClassLoader works in Java  When JVM request for a class, it invokes a loadClass() method of the java.lang.ClassLoader class by passing the fully classified name of the class. The loadClass() method calls for findLoadedClass() method to check that the class has been already loaded or not. It is required to avoid loading the class multiple times.  If the class is already loaded, it delegates the request to parent ClassLoader to load the class. If the ClassLoader is not finding the class, it invokes the findClass() method to look for the classes in the file system. The following diagram shows how ClassLoader loads class in Java using delegation.  ClassLoader in Java  Suppose that we have an application specific class Demo.class. The request for loading of this class files transfers to Application ClassLoader. It delegates to its parent Extension ClassLoader. Further, it delegates to Bootstrap ClassLoader. Bootstrap search that class in rt.jar and since that class is not there. Now request transfer to Extension ClassLoader which searches for the directory jre/lib/ext and tries to locate this class there. If the class is found there, Extension ClassLoader loads that class. Application ClassLoader never loads that class. When the extension ClassLoader does not load it, then Application ClaasLoader loads it from CLASSPATH in Java.  Visibility principle states that child ClassLoader can see the class loaded by the parent ClassLoader, but vice versa is not true. It means if Application ClassLoader loads Demo.class, in such case, trying to load Demo.class explicitly using Extension ClassLoader throws java.lang.ClassNotFoundException.  According to the uniqueness principle, a class loaded by the parent should not be loaded by Child ClassLoader again. So, it is possible to write class loader which violates delegation and uniqueness principles and loads class by itself.  In short, class loader follows the following rule:   * It checks if the class is already loaded. * If the class is not loaded, ask parent class loader to load the class. * If parent class loader cannot load class, attempt to load it in this class loader.   Consider the following Example:   1. **public** **class** Demo 2. { 3. **public** **static** **void** main(String args[]) 4. { 5. System.out.println("How are you?"); 6. } 7. }   Compile and run the above code by using the following command:   1. javac Demo.java 2. java -verbose:**class** Demo   **-verbose:class:** It is used to display the information about classes being loaded by JVM. It is useful when using class loader for loading classes dynamically. The following figure shows the output.  ClassLoader in Java  We can observe that runtime classes required by the application class (Demo) are loaded first.  When classes are loaded  There are only two cases:   * When the new byte code is executed. * When the byte code makes a static reference to a class. For example, **System.out**.   Static vs. Dynamic Class Loading  Classes are statically loaded with "new" operator. Dynamic class loading invokes the functions of a class loader at run time by using Class.forName() method.  Difference between loadClass() and Class.forName()  The loadClass() method loads only the class but does not initialize the object. While Class.forName() method initialize the object after loading it. For example, if you are using ClassLoader.loadClass() to load the JDBC driver, class loader does not allow to load JDBC driver.  The java.lang.Class.forName() method returns the Class Object coupled with the class or interfaces with the given string name. It throws ClassNotFoundException if the class is not found.  Example  In this example, java.lang.String class is loaded. It prints the class name, package name, and the names of all available methods of String class. We are using Class.forName() in the following example.  **Class<?>:** Represents a Class object which can be of any type (? is a wildcard). The Class type contains meta-information about a class. For example, type of String.class is Class<String>. Use Class<?> if the class being modeled is unknown.  **getDeclaredMethod():** Returns an array containing Method objects reflecting all the declared methods of the class or interface represented by this Class object, including public, protected, default (package) access, and private methods, but excluding inherited methods.  **getName():** It returns the method name represented by this Method object, as a String.   1. **import** java.lang.reflect.Method; 2. **public** **class** ClassForNameExample 3. { 4. **public** **static** **void** main(String[] args) 5. { 6. **try** 7. { 8. Class<?> cls = Class.forName("java.lang.String"); 9. System.out.println("Class Name: " + cls.getName()); 10. System.out.println("Package Name: " + cls.getPackage()); 11. Method[] methods = cls.getDeclaredMethods(); 12. System.out.println("-----Methods of String class -------------"); 13. **for** (Method method : methods) 14. { 15. System.out.println(method.getName()); 16. } 17. } 18. **catch** (ClassNotFoundException e) 19. { 20. e.printStackTrace(); 21. } 22. } 23. }   **Output**  Class Name: java.lang.String  Package Name: package java.lang  -----Methods of String class -------------  value  coder  equals  length  toString  hashCode  getChars  ------  ------  ------  intern  isLatin1  checkOffset  checkBoundsOffCount  checkBoundsBeginEnd  access$100  access$200 |

# Java Runtime class

**Java Runtime** class is used to interact with java runtime environment. Java Runtime class provides methods to execute a process, invoke GC, get total and free memory etc. There is only one instance of java.lang.Runtime class is available for one java application.

The **Runtime.getRuntime()** method returns the singleton instance of Runtime class.

# Daemon Thread in Java

**Daemon thread in java** is a service provider thread that provides services to the user thread. Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates this thread automatically.

There are many java daemon threads running automatically e.g. gc, finalizer etc.

You can see all the detail by typing the jconsole in the command prompt. The jconsole tool provides information about the loaded classes, memory usage, running threads etc.

## Points to remember for Daemon Thread in Java

* It provides services to user threads for background supporting tasks. It has no role in life than to serve user threads.
* Its life depends on user threads.
* It is a low priority thread.
* Methods for Java Daemon thread by Thread class
* The java.lang.Thread class provides two methods for java daemon thread.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method** | **Description** |
| 1) | public void setDaemon(boolean status) | is used to mark the current thread as daemon thread or user thread. |
| 2) | public boolean isDaemon() | is used to check that current is daemon. |

#### **Note: If you want to make a user thread as Daemon, it must not be started otherwise it will throw IllegalThreadStateException.**

### Why JVM terminates the daemon thread if there is no user thread?

The sole purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

**class** parents

{

**int** x=10;

**public** **static** **void** app()

{

System.***out***.println("static method parnts class");

}

}

**public** **class** Child **extends** parents {

**int** x=10;

**public** **static** **void** app()

{

System.***out***.println("static method parnts child class ");

}

**public** **static** **void** main(String[] args) {

parents s=**new** Child();

System.***out***.println(s.x);

parents.*app*();

}

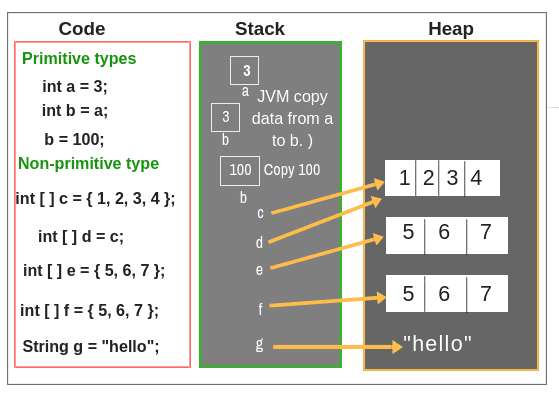
}

Output

10

static method parnts class

[](https://www.javatpoint.com/adobe-after-effect)

[](https://www.javatpoint.com/adobe-dreamweaver)

## ****Why and how to use Wait() and Notify() in Java?****

You should use Wait and Notify in [Java](https://www.edureka.co/blog/java-tutorial/) because they are related to lock and object has a lock. Even though wait and notify in Java are quite a fundamental concept, they are defined in the [object class](https://www.edureka.co/blog/java-objects-and-classes/).  Surprisingly, it’s not that easy to write the code using wait and notify. You can test this by writing code to solve the producer-consumer problem using wait and notify.

**Note**: Wait and Notify methods are defined in the object class, and they must be called inside synchronized block.

public class Thread1

{

public static void main(String[] args)

{

Thread2 b = new Thread2();

b.start();

synchronized(b)

{

try

{

System.out.println("Waiting for 2 to complete...");

b.wait();

}

catch(InterruptedException e)

{

e.printStackTrace();

}

System.out.println("Total is: " + b.total);

} } }

class Thread2 extends Thread1

{

int total;

@Override public void run()

{

synchronized(this)

{

for(int i=0; i<=100 ; i++)

{

total += i;

}

notify();

}}}

Notice that in the above example, an object of Thread2, that is b, is synchronized. This b completes the calculation before the Main thread outputs its total value.

**Output:**

Output- Wait and Notify in Java-Edureka

**Runtime Polymorphism with Data Members**

In Java, we can override methods only, not the variables(data members), so **runtime polymorphism cannot be achieved by data members.** For example :

|  |
| --- |
| // Java program to illustrate the fact that  // runtime polymorphism cannot be achieved  // by data members    // class A  class A  {      int x = 10;  }    // class B  class B extends A  {      int x = 20;  }    // Driver class  public class Test  {      public static void main(String args[])      {          A a = new B(); // object of type B            // Data member of class A will be accessed          System.out.println(a.x);      }  } |

Output:

10

**Explanation :** In above program, both the class A(super class) and B(sub class) have a common variable ‘x’. Now we make object of class B, referred by ‘a’ which is of type of class A. Since variables are not overridden, so the statement “a.x” will **always** refer to data member of super class.

[**Race conditions**](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html) occur only if multiple threads are accessing the same resource, **and** one or more of the threads **write** to the resource. If multiple threads read the same resource [**race conditions**](http://tutorials.jenkov.com/java-concurrency/race-conditions-and-critical-sections.html) do not occur.

## Thread Deadlock

A deadlock is when two or more threads are blocked waiting to obtain locks that some of the other threads in the deadlock are holding. Deadlock can occur when multiple threads need the same locks, at the same time, but obtain them in different order.

| start() | run() |
| --- | --- |
| Creates a new thread and the run() method is executed on the newly created thread. | No new thread is created and the run() method is executed on the calling thread itself. |
| Can’t be invoked more than one time otherwise throws *java.lang.IllegalStateException* | Multiple invocation is possible |
| Defined in *java.lang.Thread* class. | Defined in *java.lang.Runnable* interface and must be overriden in the implementing class. |

A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)
5. Terminated

|  |
| --- |
| 1. 1) New 2. The thread is in new state if you create an instance of Thread class but before the invocation of start() method. |

2) Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

3) Running

The thread is in running state if the thread scheduler has selected it.

4) Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

5) Terminated

A thread is in terminated or dead state when its run() method exits.

# Priority of a Thread (Thread Priority):

|  |
| --- |
| Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses. |

## 3 constants defined in Thread class:

|  |
| --- |
| 1. public static int MIN\_PRIORITY 2. public static int NORM\_PRIORITY 3. public static int MAX\_PRIORITY |

|  |
| --- |
| Default priority of a thread is 5 (NORM\_PRIORITY). The value of MIN\_PRIORITY is 1 and the value of MAX\_PRIORITY is 10. |

The join() method

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

Syntax:

|  |
| --- |
| public void join()throws InterruptedException |
| public void join(long milliseconds)throws InterruptedException |

# Time Complexity of Sorting Algorithms

We might have come across various instances where we need to process the data in a specific format without taking any further delay and the same in case of unsorted data processed with higher speed so that results could be put to some use. In such instances, we use sorting algorithms so that the desired efficiency is achieved. In this article, we will discuss various types of sorting algorithms with higher emphasis on time complexities. But, before moving any further, let's understand what complexity is and what's so important to talk about it.

### Complexity

Complexity has no formal definition at all. It just defines the rate of efficiency at which a task is executed. In data structures and algorithms, there are two types of complexities that determine the efficiency of an algorithm. They are:

**Space Complexity:** Space complexity is the total memory consumed by the program for its execution.

**Time Complexity:** It is defined as the times in number instruction, in particular, is expected to execute rather than the total time is taken. Since time is a dependent phenomenon, time complexity may vary on some external factors like processor speed, the compiler used, etc.

6.2M

172

C++ vs Java

In computer science, the time complexity of an algorithm is expressed in big O notation. Let's discuss some time complexities.

**O(1):** This denotes the constant time. 0(1) usually means that an algorithm will have constant time regardless of the input size. **Hash Maps** are perfect examples of constant time.

**O(log n):** This denotes logarithmic time. O(log n) means to decrease with each instance for the operations. **Binary search trees** are the best examples of logarithmic time.

**O(n):** This denotes linear time. O(n) means that the performance is directly proportional to the input size. In simple terms, the number of inputs and the time taken to execute those inputs will be proportional or the same. Linear search in **arrays** is the best example of linear time complexity.

**O(n2):** This denotes quadratic time. O(n2) means that the performance is directly proportional to the square of the input taken. In simple, the time taken for execution will take square times the input size. **Nested loops** are perfect examples of quadratic time complexity.

Let's move on to the main plan and discuss the time complexities of different sorting algorithms.

### Time Complexity of Bubble Sort

Bubble sort is a simple sorting algorithm where the elements are sorted by comparing each pair of elements and switching them if an element doesn't follow the desired order of sorting. This process keeps repeating until the required order of an element is reached.

Average case time complexity: **O(n2)**

Worst-case time complexity: **O(n2)**

Best case time complexity: **O(n)**

The best case is when the given list of elements is already found sorted. This is why bubble sort is not considered good enough when the input size is quite large.

### Time Complexity of Selection Sort

Selection sort works on the fundamental of in-place comparison. In this algorithm, we mainly pick up an element and move on to its correct position. This process is carried out as long as all of them are sorted in the desired order.

Average case time complexity: **O(n2)**

Worst-case time complexity: **O(n2)**

Best case time complexity: **O(n2)**

Selection sort also suffers the same disadvantage as we saw in the bubble sort. It is inefficient to sort large data sets. It is usually preferred because of its simplicity and performance-enhancing in situations where auxiliary memory is limited.

### Time Complexity of Insertion Sort

Insertion sort works on the phenomenon by taking inputs and placing them in the correct order or location. Thus, it is based on iterating over the existing elements while taking input and placing them where they are ought to be.

Best case time complexity: **O(n)**

Average and worst-case time complexity: **O(n2)**

**Time Complexity of QuickSort**

Quicksort works under the hood of the famous divide and conquer algorithm. In this technique, large input arrays are divided into smaller sub-arrays, and these sub-arrays are recursively sorted and merged into an enormous array after sorting.

Best and Average time complexity: **O(n log n)**

Worst-case time complexity: **(n2)**

### Time Complexity Of Merge Sort

Merge Sort also works under the influence of the divide and conquer algorithm. In this sorting technique, the input array is divided into half, and then these halves are sorted. After sorting, these two halved sub-arrays are merged into one to form a complete sorted array.

Best and Average time complexity: **O(n log n)**

Worst-case time complexity: **O(n log n)**

## Conclusion

Time complexity plays a crucial role in determining the overall performance of a program. It is solely intended to improve the performance of a program and impact the overall performance of the system. However, with great speed comes greater responsibility. Hence, to achieve the best time complexity, a developer needs to have a keen eye on using a particular algorithm or technique that delivers the best case complexity. Furthermore, to be at such a pace, a developer needs to carry prior knowledge about the sorting algorithm. Therefore, it is highly recommended to understand each of the techniques discussed in this article in detail and figure out the best one that suits the situation.

How to create thread

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:

|  |
| --- |
| Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface. |

Commonly used Constructors of Thread class:

|  |
| --- |
| * Thread() * Thread(String name) * Thread(Runnable r) * Thread(Runnable r,String name) |

Commonly used methods of Thread class:

|  |
| --- |
| 1. **public void run():**is used to perform action for a thread. 2. **public void start():**starts the execution of the thread.JVM calls the run() method on the thread. 3. **public void sleep(long miliseconds):**Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds. 4. **public void join():**waits for a thread to die. 5. **public void join(long miliseconds):**waits for a thread to die for the specified miliseconds. 6. **public int getPriority():**returns the priority of the thread. 7. **public int setPriority(int priority):**changes the priority of the thread. 8. **public String getName():**returns the name of the thread. 9. **public void setName(String name):**changes the name of the thread. 10. **public Thread currentThread():**returns the reference of currently executing thread. 11. **public int getId():**returns the id of the thread. 12. **public Thread.State getState():**returns the state of the thread. 13. **public boolean isAlive():**tests if the thread is alive. 14. **public void yield():**causes the currently executing thread object to temporarily pause and allow other threads to execute. 15. **public void suspend():**is used to suspend the thread(depricated). 16. **public void resume():**is used to resume the suspended thread(depricated). 17. **public void stop():**is used to stop the thread(depricated). 18. **public boolean isDaemon():**tests if the thread is a daemon thread. 19. **public void setDaemon(boolean b):**marks the thread as daemon or user thread. 20. **public void interrupt():**interrupts the thread. 21. **public boolean isInterrupted():**tests if the thread has been interrupted. 22. **public static boolean interrupted():**tests if the current thread has been interrupted. |

Runnable interface:

|  |
| --- |
| The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run(). |

|  |
| --- |
| 1. **public void run():**is used to perform action for a thread. |

Starting a thread:

|  |
| --- |
| **start() method** of Thread class is used to start a newly created thread. It performs following tasks:   * A new thread starts(with new callstack). * The thread moves from New state to the Runnable state. * When the thread gets a chance to execute, its target run() method will run. |

1) Java Thread Example by extending Thread class

1. **class** Multi **extends** Thread{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
5. **public** **static** **void** main(String args[]){
6. Multi t1=**new** Multi();
7. t1.start();
8. }
9. }

Output:thread is running...

2) Java Thread Example by implementing Runnable interface

1. **class** Multi3 **implements** Runnable{
2. **public** **void** run(){
3. System.out.println("thread is running...");
4. }
6. **public** **static** **void** main(String args[]){
7. Multi3 m1=**new** Multi3();
8. Thread t1 =**new** Thread(m1);
9. t1.start();
10. }
11. }

Output:thread is running...

|  |
| --- |
| If you are not extending the Thread class,your class object would not be treated as a thread object.So you need to explicitely create Thread class object.We are passing the object of your class that implements Runnable so that your class run() method may execute. |

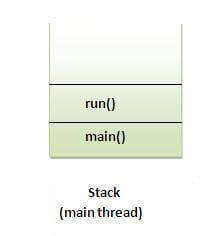
# What if we call run() method directly instead start() method?

|  |
| --- |
| * Each thread starts in a separate call stack. * Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack. |

1. **class** TestCallRun1 **extends** Thread{
2. **public** **void** run(){
3. System.out.println("running...");
4. }
5. **public** **static** **void** main(String args[]){
6. TestCallRun1 t1=**new** TestCallRun1();
7. t1.run();//fine, but does not start a separate call stack
8. }
9. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestCallRun1)

Output:running...

 ***Problem if you direct call run() method***

1. **class** TestCallRun2 **extends** Thread{
2. **public** **void** run(){
3. **for**(**int** i=1;i<5;i++){
4. **try**{Thread.sleep(500);}**catch**(InterruptedException e){System.out.println(e);}
5. System.out.println(i);
6. }
7. }
8. **public** **static** **void** main(String args[]){
9. TestCallRun2 t1=**new** TestCallRun2();
10. TestCallRun2 t2=**new** TestCallRun2();
12. t1.run();
13. t2.run();
14. }
15. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestCallRun2)

Output:1

2

3

4

5

1

2

3

4

5

|  |
| --- |
| As you can see in the above program that there is no context-switching because here t1 and t2 will be treated as normal object not thread object. |

# Difference between Runnable and Callable interface in java

[Java](https://www.tutorialspoint.com/questions/category/Java)[Server Side Programming](https://www.tutorialspoint.com/questions/category/Server-Side-Programming)[Programming](https://www.tutorialspoint.com/questions/category/Programming)

Runnable and Callable both functional interface. Classes which are implementing these interfaces are designed to be executed by another thread.

Thread can be started with Ruunable and they are two ways to start a new thread: one is by subclassing Thread class and another is implementing Runnable interface.

Thread class does not have constructor for callable so we should use ExecutorService  class for executing thread.

| **Sr. No.** | **Key** | **Runnable** | **Callable** |
| --- | --- | --- | --- |
| 1 | Package | It belongs to Java.lang | It belongs to java.util.concurrent |
| 2 | Thread Creation | We can create thread by passing runnable as a parameter. | We can’t create thread by passing callable as parameter |
| 3 | Return Type | Ruunable does not return anything | Callable can return results |
| 4. | Method | It has run() method | It has call()method |
| 5 | Bulk Execution | It can’t be used for bulk execution of task | It can be used for bulk execution of task by invoking invokeAll(). |

### Example of Runnable

public class RunnableExample implements Runnable {

   public void run() {

      System.out.println("Hello from a Runnable!");

   }

   public static void main(String args[]) {

      (new Thread(new RunnableExample())).start();

   }

}

### Example of Callable

public class Main {

   public static void main(String args[]) throws InterruptedException, ExecutionException {

      ExecutorService services = Executors.newSingleThreadExecutor();

      Future<?> future = services.submit(new Task());

      System.out.println("In Future Object" + future.get());

   }

}

import java.util.concurrent.Callable;

public class Task implements Callable {

   @Override

   public String call() throws Exception {

      System.out.println("In call");

      String name = "test";

      return name;

   }

}

# Deadlock in java

Deadlock in java is a part of multithreading. Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread. Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



### Example of Deadlock in java

1. **public** **class** TestDeadlockExample1 {
2. **public** **static** **void** main(String[] args) {
3. **final** String resource1 = "ratan jaiswal";
4. **final** String resource2 = "vimal jaiswal";
5. // t1 tries to lock resource1 then resource2
6. Thread t1 = **new** Thread() {
7. **public** **void** run() {
8. **synchronized** (resource1) {
9. System.out.println("Thread 1: locked resource 1");
11. **try** { Thread.sleep(100);} **catch** (Exception e) {}
13. **synchronized** (resource2) {
14. System.out.println("Thread 1: locked resource 2");
15. }
16. }
17. }
18. };
20. // t2 tries to lock resource2 then resource1
21. Thread t2 = **new** Thread() {
22. **public** **void** run() {
23. **synchronized** (resource2) {
24. System.out.println("Thread 2: locked resource 2");
26. **try** { Thread.sleep(100);} **catch** (Exception e) {}
28. **synchronized** (resource1) {
29. System.out.println("Thread 2: locked resource 1");
30. }
31. }
32. }
33. };

36. t1.start();
37. t2.start();
38. }
39. }

Output: Thread 1: locked resource 1

Thread 2: locked resource 2

* Difference between start() and run() method in Java? ([answer](http://www.java67.com/2015/12/difference-between-thread-start-and-run-method-java.html))
* How to join multiple threads in Java? ([answer](http://javarevisited.blogspot.com/2013/02/how-to-join-multiple-threads-in-java-example-tutorial.html))
* How to stop a thread in Java? ([answer](http://javarevisited.blogspot.com/2011/10/how-to-stop-thread-java-example.html))
* Top 50 Thread Interview Questions for experienced programmers ([list](http://javarevisited.blogspot.com/2014/07/top-50-java-multithreading-interview-questions-answers.html))
* How to avoid deadlock in Java? ([answer](https://javarevisited.blogspot.com/2018/08/how-to-avoid-deadlock-in-java-threads.html))
* 10 Free Java Courses for Beginners and Intermediate developers ([courses](https://www.java67.com/2018/08/top-10-free-java-courses-for-beginners-experienced-developers.html))
* How Happens Before works in Java?  ([answer](https://javarevisited.blogspot.com/2020/01/what-is-happens-before-in-java-concurrency.html))
* 10 Java Multithreading and Concurrency Best Practices ([article](https://javarevisited.blogspot.com/2015/05/top-10-java-multithreading-and.html#axzz5Neevu8QO))
* Understanding the flow of data and code in Java program ([answer](https://javarevisited.blogspot.com/2019/02/thread-code-and-data-how-multithreading-java-program-execute.html))
* Top 5 Books to Master Concurrency in Java ([books](https://javarevisited.blogspot.com/2016/06/5-books-to-learn-concurrent-programming-multithreading-java.html))
* Is Java Concurrency in Practice still valid in 2021 ([answer](https://javarevisited.blogspot.com/2018/07/is-java-concurrency-in-practice-still-relevant-in-era-of-java8.html))
* Difference between CyclicBarrier and CountDownLatch in Java? ([answer](http://www.java67.com/2012/08/difference-between-countdownlatch-and-cyclicbarrier-java.html))
* 10 Tips to become a better Java Developer in 2021 ([tips](https://javarevisited.blogspot.com/2018/05/10-tips-to-become-better-java-developer.html#axzz61Tq0rRG1))
* How to solve the producer-consumer problem using a blocking queue in Java? ([solution](http://www.java67.com/2015/12/producer-consumer-solution-using-blocking-queue-java.html))
* How to do inter-thread communication in Java using wait-notify? ([answer](https://javarevisited.blogspot.com/2013/12/inter-thread-communication-in-java-wait-notify-example.html))
* Top 5 Courses to Learn Java Multithreading in-depth ([courses](https://javarevisited.blogspot.com/2018/06/top-5-java-multithreading-and-concurrency-courses-experienced-programmers.html))

# How to create an immutable class with mutable object references in Java?

**Immutable objects** are those objects whose states **cannot be changed once initialized**. Sometimes it is necessary to make an immutable class as per the requirement. For example, All **primitive wrapper classes** (Integer, Byte, Long, Float, Double, Character, Boolean and Short) are immutable in Java. **String class** is also an immutable class.

**To create a custom immutable class we have to do the following steps**

* Declare the class as **final** so it can’t be extended.
* Make all**fields private** so that direct access is not allowed.
* Do not provide **setter methods** (methods that modify fields) for variables, so that it can not be set.
* Make all **mutable fields final** so that their values can be assigned only once.
* Initialize all the fields through a **constructor** doing the deep copy.
* Perform **cloning of objects** in the getter methods to return a copy rather than returning the actual object reference.
* If the instance fields include **references to mutable objects**, don’t allow those objects to be changed
* Don’t provide methods that modify the **mutable** **objects**.
* Don’t share **references** **to the mutable objects**. Never store references to external, mutable objects passed to the constructor. If necessary, create copies and store references to the copies. Similarly, create copies of our internal mutable objects when necessary to avoid returning the originals in our methods.
* // Employee.java
* final class Employee {
* private final String empName;
* private final int age;
* private final Address address;
* public Employee(String name, int age, Address address) {
* super();
* this.empName = name;
* this.age = age;
* this.address = address;
* }
* public String getEmpName() {
* return empName;
* }
* public int getAge() {
* return age;
* }
* /\* public Address getAddress() {
* return address;
* }
* \*/
* public Address getAddress() throws CloneNotSupportedException {
* return (Address) address.clone();
* }
* }
* // Address.java
* class Address implements Cloneable {
* public String addressType;
* public String address;
* public String city;
* public Address(String addressType, String address, String city) {
* super();
* this.addressType = addressType;
* this.address = address;
* this.city = city;
* }
* public String getAddressType() {
* return addressType;
* }
* public void setAddressType(String addressType) {
* this.addressType = addressType;
* }
* public String getAddress() {
* return address;
* }
* public void setAddress(String address) {
* this.address = address;
* }
* public String getCity() {
* return city;
* }
* public void setCity(String city) {
* this.city = city;
* }
* public Object clone() throws CloneNotSupportedException {
* return super.clone();
* }
* @Override
* public String toString() {
* return "Address Type - "+addressType+", address - "+address+", city - "+city;
* }
* }
* // MainClass.java
* public class MainClass {
* public static void main(String[] args) throws Exception {
* Employee emp = new Employee("Adithya", 34, new Address("Home", "Madhapur", "Hyderabad"));
* Address address = emp.getAddress();
* System.out.println(address);
* address.setAddress("Hi-tech City");
* address.setAddressType("Office");
* address.setCity("Hyderabad");
* System.out.println(emp.getAddress());
* }
* }
* In the above example, instead of returning the original **Address** object we will return a **deep cloned copy** of that instance. The address class must implement the **Cloneable** interface.

## Output

* Address Type - Home, address - Madhapur, city - Hyderabad
* Address Type - Home, address - Madhapur, city - Hyderabad

--------------threading ----------------------------

**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.

**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  implementingjava.lang.Runnableinterface.

We will create object of class which implements Runnable interface :

MyRunnable runnable=new MyRunnable();

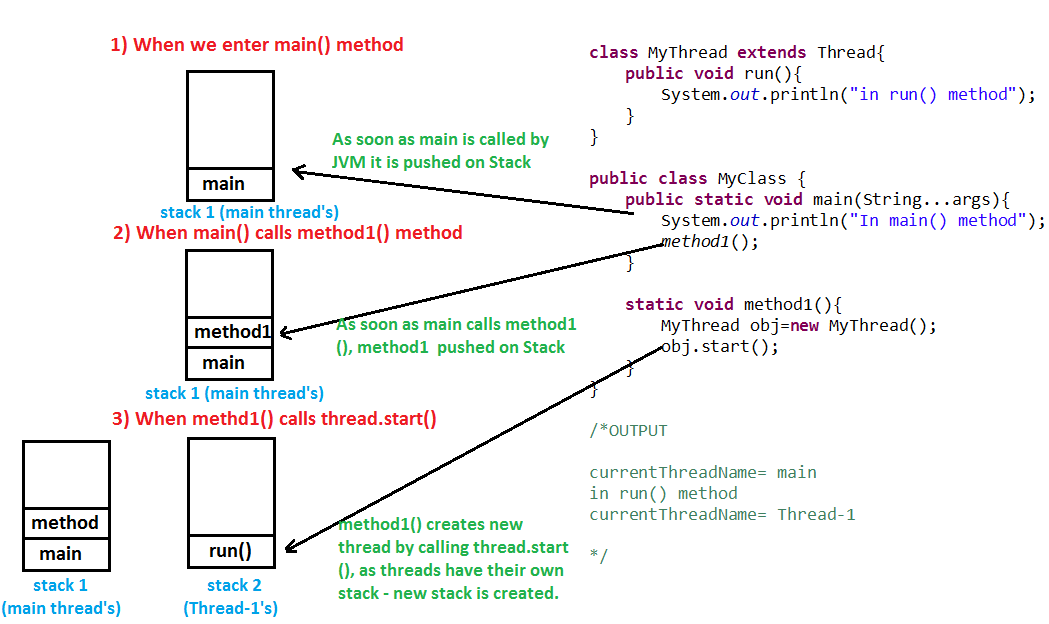
Thread thread=new Thread(runnable);

 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.

**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.

volatile void method1(){} //it’s illegal, compilation error.

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:

synchronized void 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.  .
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() 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)

**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.

**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

public void 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).

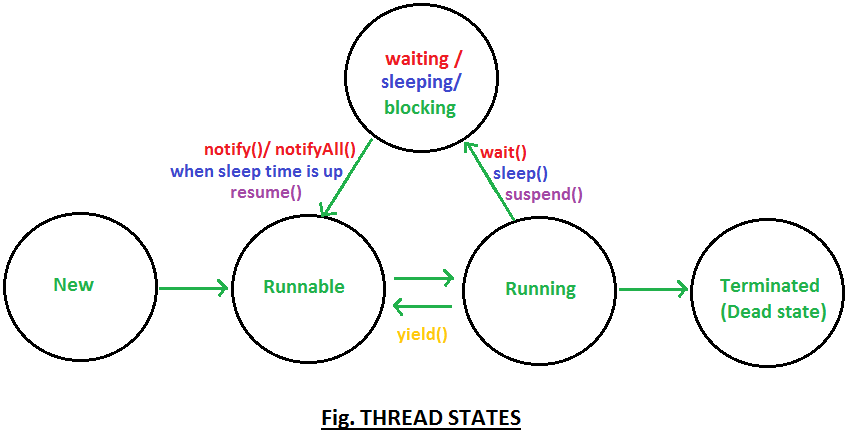
**Question 22. What is life cycle of Thread, explain thread states? (Important)**

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.

**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)**?**

**A**nswer.[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 -

|  |
| --- |
| public static native void 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, it’s 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

|  |
| --- |
| public static native void 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.

|  |
| --- |
| public static native void 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.
* Belongs to which class :[sleep()](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() methodnever 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.

public class 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.

public class MyClass {

final inti=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.

public class MyClass {

void method(){

inti=0; //Local variable, is thread safe.

}

}

1. Using thread safe collections : Rather than using ArrayList we must Vector and in place of using HashMap we must use ConcurrentHashMap or HashTable.
2. 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.
3. 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.
4. 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)

|  |
| --- |
| public final native void 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=newMyclass();   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.

**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 tillproductionInProcess 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");

}

**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 begunnew 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);

}

});

**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.

**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. \*/

publicfinalstaticintMIN\_PRIORITY= 1;

/\* The default priority that is assigned to a thread. \*/

publicfinalstaticintNORM\_PRIORITY= 5;

/\* The maximum priority that a thread can have. \*/

publicfinalstaticintMAX\_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.

Topics:

Opinions expressed by DZone contributors are their own.

### Popular on DZone

* [**Ping-Pong Implementation: JSR-356**](https://dzone.com/articles/ping-pong-implementation-jsr-356?fromrel=true)
* [**Cloud factory - Architectural introduction**](https://dzone.com/articles/cloud-factory-architectural-introduction?fromrel=true)
* [**Understanding Jakarta EE 8 - C.D.I. Part 3 — Defining Custom CDI Annotations**](https://dzone.com/articles/understanding-jakarta-ee-8-cdi-part-3-defining-cus?fromrel=true)
* [**Skipping Retrospectives?**](https://dzone.com/articles/skipping-retrospectives?fromrel=true)

## Java Partner Resources

AbstractionEncapsulationAbstraction is the process or method of gaining the information.While encapsulation is the process or method to contain the information.In abstraction, problems are solved at the design or interface level.While in encapsulation, problems are solved at the implementation level.Abstraction is the method of hiding the unwanted information.Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside.We can implement abstraction using abstract class and interfaces.Whereas encapsulation can be implemented using by access modifier i.e. private, protected and public.In abstraction, implementation complexities are hidden using abstract classes and interfaces.While in encapsulation, the data is hidden using methods of getters and setters.The objects that help to perform abstraction are encapsulated.Whereas the objects that result in encapsulation need not be abstracted.

public class Test{

public static void main(String[] arr){

System.out.println(0.1\*3 == 0.3);

System.out.println(0.1\*2 == 0.2);

}

}

class MyThread extends Thread

{

public void run()

{

System.out.println("Running");

}

}

class ThreadTest {

public static void main(String args[]) throws InterruptedException

{

Runnable r = new MyThread(); // #1

Thread myThread = new Thread(r); // #2

myThread.start();

}

}