JAVA BASICS

4KCJ001 - LECTURE 7

By
Sudha Agarwal
sudha.agarwal@4kitsolutions.com

http://www.4kitsolutions.com/

INDEX

❖ MultiThreading

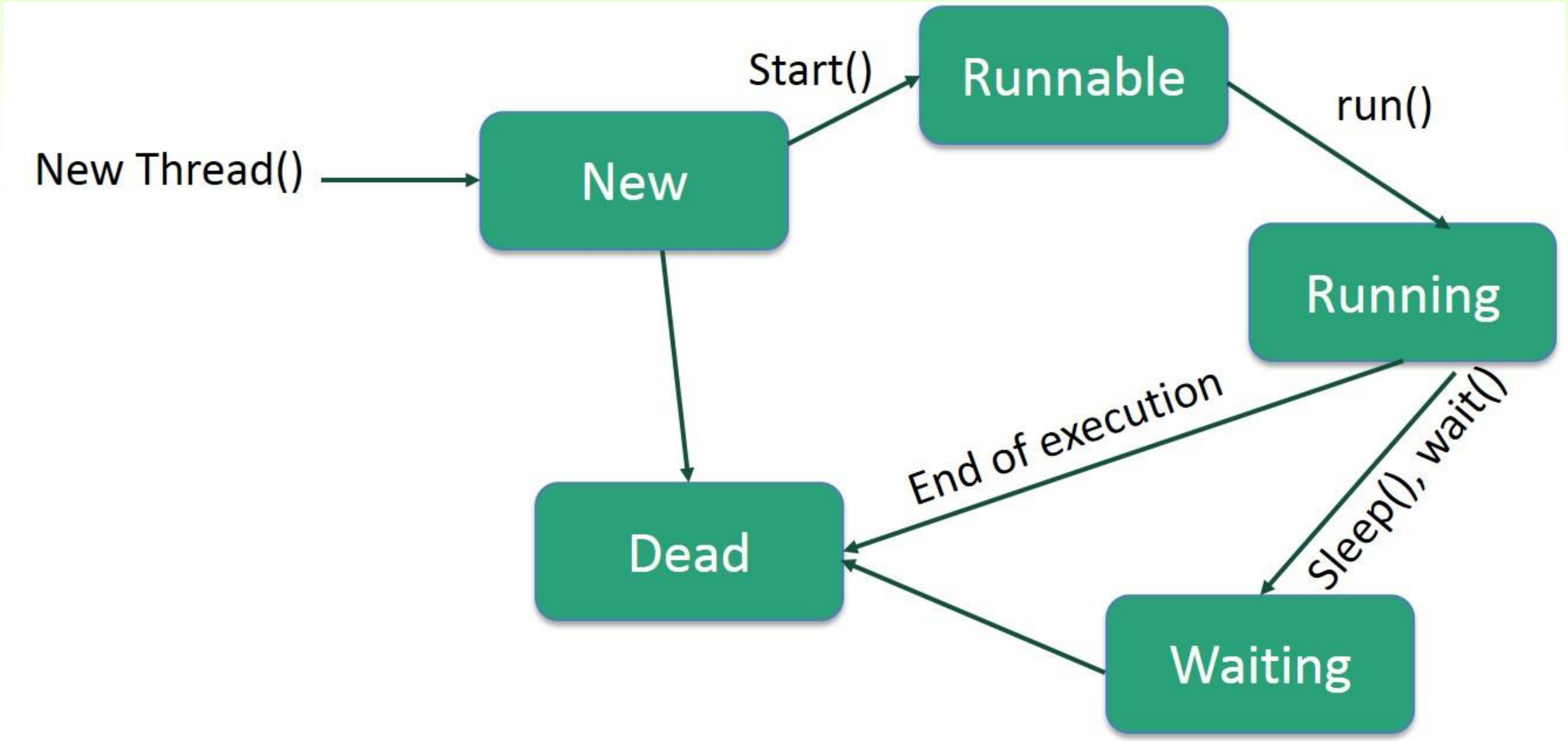
MULTITHREADING

❖Java is a multi-threaded programming language which means we can develop multi-threaded program using Java.

❖A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

❖Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

LIFE CYCLE OF A THREAD



http://www.4kitsolutions.com/

LIFE CYCLE OF A THREAD

❖Following are the stages of the life cycle −

❖**New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

❖**Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.

❖**Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.

LIFE CYCLE OF A THREAD

❖**Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.

❖**Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

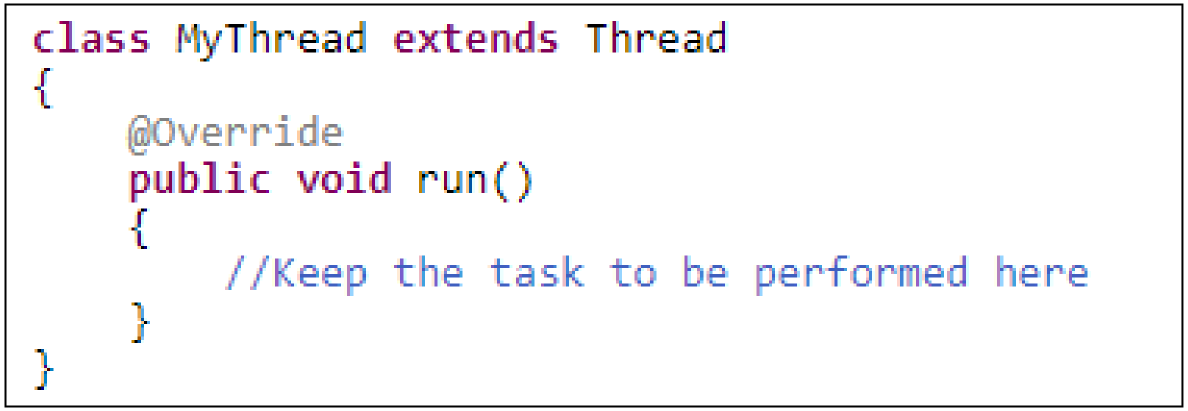
CREATING THREADS

- ❖There are two ways to create threads in java language:
- ❖1) By extending java.lang.Thread class.
- ❖2) By implementing **java.lang.Runnable** interface

EXTENDING JAVA.LANG.THREAD

❖You can create your own thread by extending **Thread** class of java.lang package.

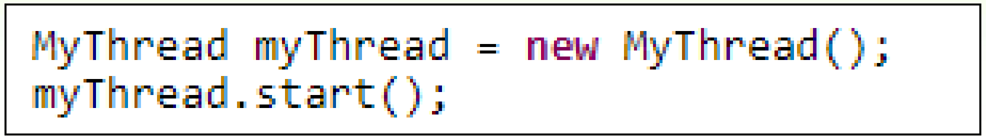
❖You have to override **run()** method of Thread class and keep the task which you want your thread to perform in this run() method. Here is the syntax of creating a thread by extending Thread class.



```
class MyThread extends Thread
{
    @Override
    public void run()
    {
        //Keep the task to be performed here
    }
}
```

EXTENDING JAVA.LANG.THREAD

❖After defining your thread, create an object of your thread and call start() method where ever you want this task to be performed.

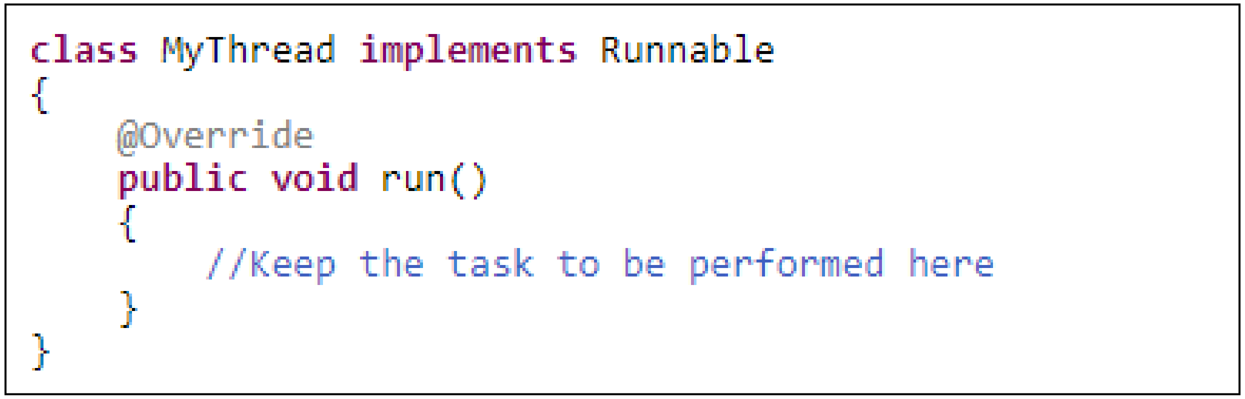


```
MyThread myThread = new MyThread();
myThread.start();
```

IMPLEMENTING JAVA.LANG.RUNNABLE

❖Another method of creating a thread is to implement **Runnable** interface of java.lang package. Runnable interface has only one abstract method i.e **run()**.

❖You have to implement this method and keep the task to be performed in this method.

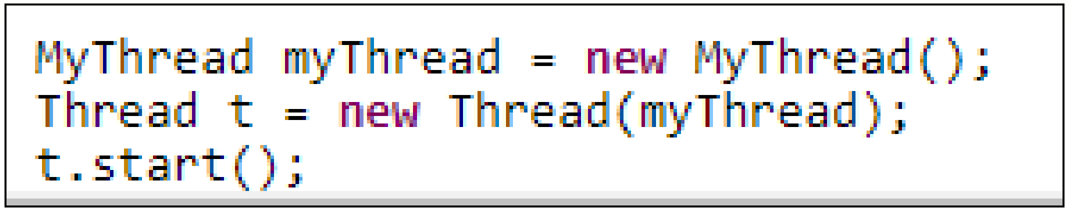


```
class MyThread implements Runnable
{
    @Override
    public void run()
    {
        //Keep the task to be performed here
    }
}
```

http://www.4kitsolutions.com/

IMPLEMENTING JAVA.LANG.RUNNABLE

❖After defining the thread, create an object to java.lang.Thread class through a constructor which takes Runnable type as an argument and pass the object of your thread that implements Runnable interface as an argument to it and call the start() method



```
MyThread myThread = new MyThread();
Thread t = new Thread(myThread);
t.start();
```

EXTENDS VS. IMPLEMENTS

❖**1) Multiple Inheritance Limitation -**

❖As you know, Java doesn't support multiple inheritance. A class in java can extend only one class. If you extend Thread class, then your class will not be able to extend any other class. This will limit your class to thread behavior. If you implement Runnable interface, then you will have an option for your class to extend any other class and inherit behaviors from other class also.

❖**2) Overhead Of Additional Methods -**

❖If you extend Thread class, all methods of Thread class will be inheriting to your class which you may not need. This will cause additional overhead. You can remove this overhead by implementing Runnable interface.

EXTENDS VS. IMPLEMENTS

❖**3) Logical Separation Of Task From The Runner –**

❖If you implement Runnable interface, it will separate actual task from the runner. Runnable interface represents only the task and you can pass this task to any type of runner, either a thread or any executors.

❖**4) Best Object Oriented Design Practice –**

❖In object oriented programming, extending a class means modifying or improving the existing class. If you are not improving the class, then it is not a good practice to extend it. So, implementing Runnable will be the best object oriented design practice.

EXTENDS VS. IMPLEMENTS

❖**5) Loosely Coupled Vs Tightly coupled –**

❖"Implements Runnable" makes your code loosely coupled. Because it separates the task from the runner. "Extends Thread" will make your code tightly coupled. Because, single class will act as both task container as well as runner.

❖**6) Reusability -**

❖Implementing Runnable improves the reusability of your code. Because, Runnable contains only the task and you can use it wherever and whenever you want.

THREAD PRIORITIES

❖Every Java thread has a priority that helps the operating system determine the order in which threads are scheduled.

❖Java thread priorities are in the range between MIN_PRIORITY (a constant of 1) and MAX_PRIORITY (a constant of 10). By default, every thread is given priority NORM_PRIORITY (a constant of 5).

❖Threads with higher priority are more important to a program and should be allocated processor time before lower-priority threads. However, thread priorities cannot guarantee the order in which threads execute and are very much platform dependent.

THREAD PRIORITIES

❖There are two methods in java.lang.Thread class related to priority of a thread.

❖They are **setPriority()** and **getPriority** methods. **setPriority()** method is used to set the priority of a thread and **getPriority()** method is used to retrieve the priority of a thread.

❖**public final void setPriority(int newPriority)** —> Changes the priority of a thread to newPriority.

❖**public final int getPriority()** —> Returns the priority of a thread.

Types of Threads in Java

- ❖There are two types of Threads in java.
- ❖1) User Thread
- ❖2) Daemon Thread

USER THREAD

❖User threads are threads which are created by the application or user. They are high priority threads.

❖JVM (Java Virtual Machine) will not exit until all user threads finish their execution. JVM wait for these threads to finish their task. These threads are foreground threads.

DAEMON THREAD

❖Daemon threads are threads which are mostly created by the JVM. These threads always run in background.

❖These threads are used to perform some background tasks like garbage collection and house-keeping tasks. These threads are less priority threads.

❖JVM will not wait for these threads to finish their execution.

❖JVM will exit as soon as all user threads finish their execution. JVM doesn’t wait for daemon threads to finish their task.

THREADS

❖You can convert user thread into daemon thread explicitly by calling setDaemon() method of the thread.

```
UserThread userThread = new UserThread();    //Creating the
UserThread

userThread.setDaemon(true);          //Changing the thread as
Daemon

userThread.start();                //Starting the thread
```

THREADS

❖You can check whether the thread is user thread or a daemon thread by using isDaemon() method of Thread class. This method returns “true” for a daemon thread and “false” for a user thread.

❖Daemon property of a thread is inherited from it’s parent thread. i.e The thread created by user thread will be user thread and the thread created by daemon thread will be a daemon thread.

❖The main thread or primary thread created by JVM is an user thread

THREAD.SLEEP()

❖Thread.sleep() method makes the currently executing thread to pause it’s execution for a specified period of time. There are two overloaded forms of sleep() method available in java.lang.Thread class.

❖1) **public static void sleep(long millis) throws InterruptedException**

❖—> It causes the currently executing thread to sleep for specified number of milliseconds.

❖2) **public static void sleep(long millis, int nanos) throws InterruptedException**

❖—> It makes the currently executing thread to sleep for specified number of milliseconds plus specified number of nanoseconds.

THREAD.SLEEP()

❖Thread.sleep() method throws InterruptedException if a thread in sleep is interrupted by other threads.

❖InterruptedException is a checked type of exception. That means, “Thread.sleep()” statement must be enclosed within try-catch blocks or it must be specified with throws clause.

JOINING THE THREADS

❖**join()** method of java.lang.Thread class is used to mantain the order of execution of threads.

❖Using join() method, you can make the currently executing thread to wait for the some other threads to finish their task.

❖For example, Let's us assume that there are two threads namely, thread1 and thread2. You can make thread1 to hold it's execution for some time so that thread2 can finish it's task. After, thread2 finishes it's task, thread1 resumes it's execution.

❖For this to happen, you should call join() method on thread2 within thread1.

JOINING THE THREADS

❖There are three forms of join() method available in Thread class.

❖**1) public final void join() throws InterruptedException**

❖—> Currently executing thread waits for a thread to finish it’s task on which it is called.

❖**2) public final void join(long millis) throws InterruptedException**

❖—> currently executing thread waits at most millis milliseconds for a thread to finish it’s task on which it is called.

❖**3) public final void join(long millis, int nanos) throws InterruptedException**

❖—> Currently executing thread waits at most millis milliseconds plus nanos nanoseconds for a thread to finish it’s task on which it is called.

SYNCHRONIZATION

❖**Synchronization** in java is a strategy or a method to avoid thread interference and hence protecting the data from inconsistency.

❖Synchronization is also one of the way to make code thread safe. Through synchronization, we can make the threads to execute particular method or block in sync not simultaneously.

❖Synchronization in java is implemented using **synchronized** keyword. synchronized keyword can be used with methods or blocks but not with the variables

SYNCHRONIZATION

❖When a method or block is declared as synchronized, only one thread can enter into that method or block.

❖When one thread is executing synchronized method or block, the other threads which wants to execute that method or block wait or suspend their execution until first thread is done with that method or block.

❖Thus avoiding the thread interference and achieving thread safeness.

INTERTHREAD COMMUNICATION

❖Threads can communicate with each other using **wait()**, **notify()** and **notifyAll()** methods.

❖These methods are final methods of java.lang.Object class. That means every class in java will have these methods.

❖These three methods must be called within synchronized method or block. Any thread which calls these methods must have lock of that object.

INTERTHREAD COMMUNICATION

❖1) **public final void wait() throws InterruptedException**

❖This method tells the currently executing thread to release the lock of this object and wait until some other thread acquires the same lock and notify it using either notify() or notifyAll() methods. This method throws InterruptedException if waiting thread is interrupted.

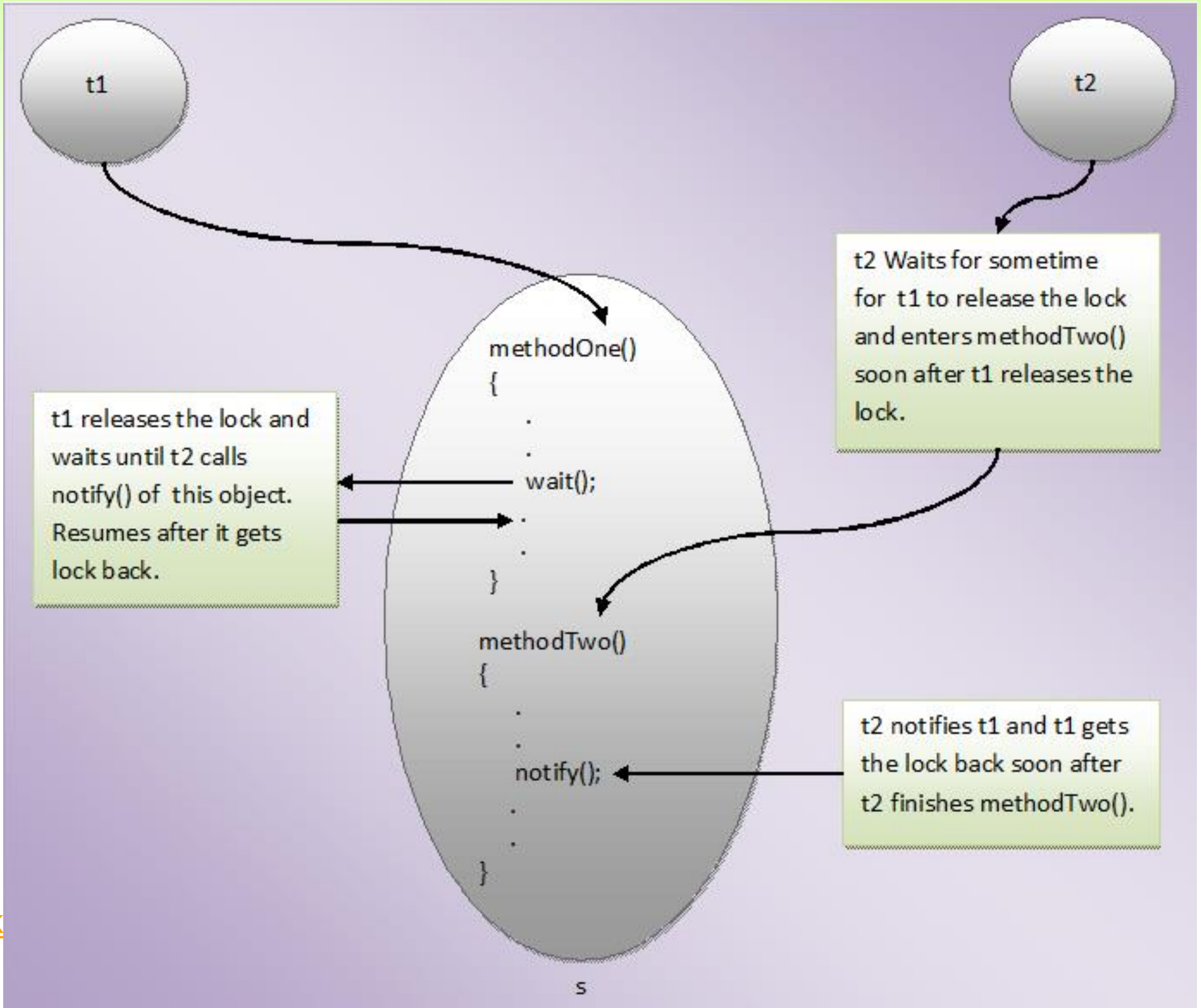
❖**2)public final void notify()**

❖This method wakes up one thread randomly that called wait() method on this object.

❖**3) public final void notifyAll()**

❖This method wakes up all the threads that called wait() method on this object. But, only one thread will acquire lock of this object depending upon the priority.

INTERTHREAD COMMUNICATION


t1
t2
t2 Waits for sometime for t1 to release the lock and enters methodTwo() soon after t1 releases the lock.
methodOne()
{
.
.
wait();
.
.
}
t1 releases the lock and waits until t2 calls notify() of this object. Resumes after it gets lock back.
methodTwo()
{
.
.
notify();
.
.
}
t2 notifies t1 and t1 gets the lock back soon after t2 finishes methodTwo().
s

INTERTHREAD COMMUNICATION

❖If a thread calls notify() method and more than one threads are waiting for the object lock, then only one thread will be notified randomly.

❖When a thread calls notifyAll() method on an object, it notifies all the threads which are waiting for this object lock. But, only one thread will acquire this object lock depending upon priority.

❖When you call sleep() method on a thread, thread goes to sleep with holding the object lock with it. But, if you call wait() method, thread releases the object lock and goes for sleep. This is the main difference between wait() and sleep() methods.

❖wait(), notify() and notifyAll() are final methods of java.lang.Object class not java.lang.Thread class.

INTERTHREAD COMMUNICATION

❖wait(), notify() and notifyAll() – all these three methods throw IllegalMonitorStateException if the calling thread does not owns the object lock.

THREAD INTERRUPTION

❖**Thread interruption** in java is a mechanism in which a thread which is either sleeping or waiting can be made to stop sleeping or waiting.

❖Thread interruption is like telling the thread that it should stop waiting or sleeping and return to running status.

❖Thread interruption is programmatically implemented using interrupt() method of java.lang.Thread class.

❖**interrupt()** method is a non-static public method of Thread class.

MORE ABOUT THREADS

❖Is the code below correct?

```
class Threads_Example
{
    public static void main(String[] args)
    {
        Thread t = new Thread();

        t.start();

        t.start();
    }
}
```

http://www.4kitsolutions.com/

MORE ABOUT THREADS

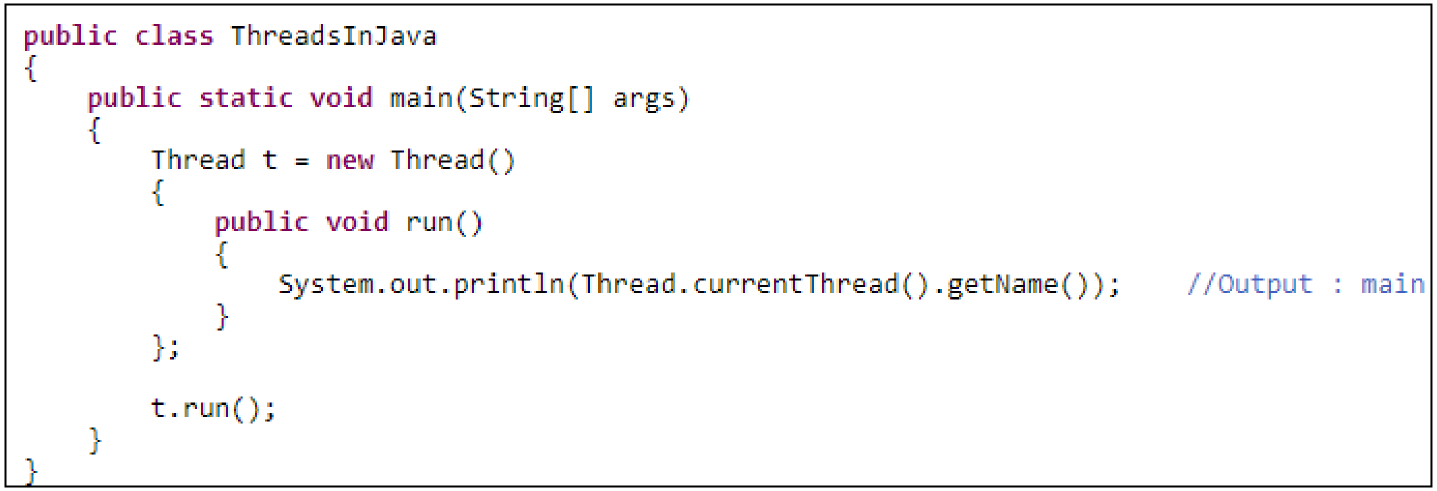
❖If you start a thread that is already started, you will get java.lang.IllegalThreadStateException at run time. There will be no compilation errors.

❖Exception is thread wise not execution wise. i.e exception effects the thread in which it occurs. Other threads will execute normally.

MORE ABOUT THREADS

❖ As we all know that start() method internally calls run() method. What happens when you call run() method directly?

❖When you call run() method of a thread directly, calling thread will execute the task defined in the run() method. In this case, main thread will execute run() method not thread t.



```
public class ThreadsInJava
{
    public static void main(String[] args)
    {
        Thread t = new Thread()
        {
            public void run()
            {
                System.out.println(Thread.currentThread().getName());    //Output : main
            }
        };

        t.run();
    }
}
```

http://www.4kitsolutions.com/

MORE ABOUT THREADS

❖ Which one is better way to implement threads in java. Is it using Thread class or using Runnable interface?

❖when multiple threads need to execute same task, then use Runnable interface. If multiple threads need to execute different tasks, then go for Thread class.

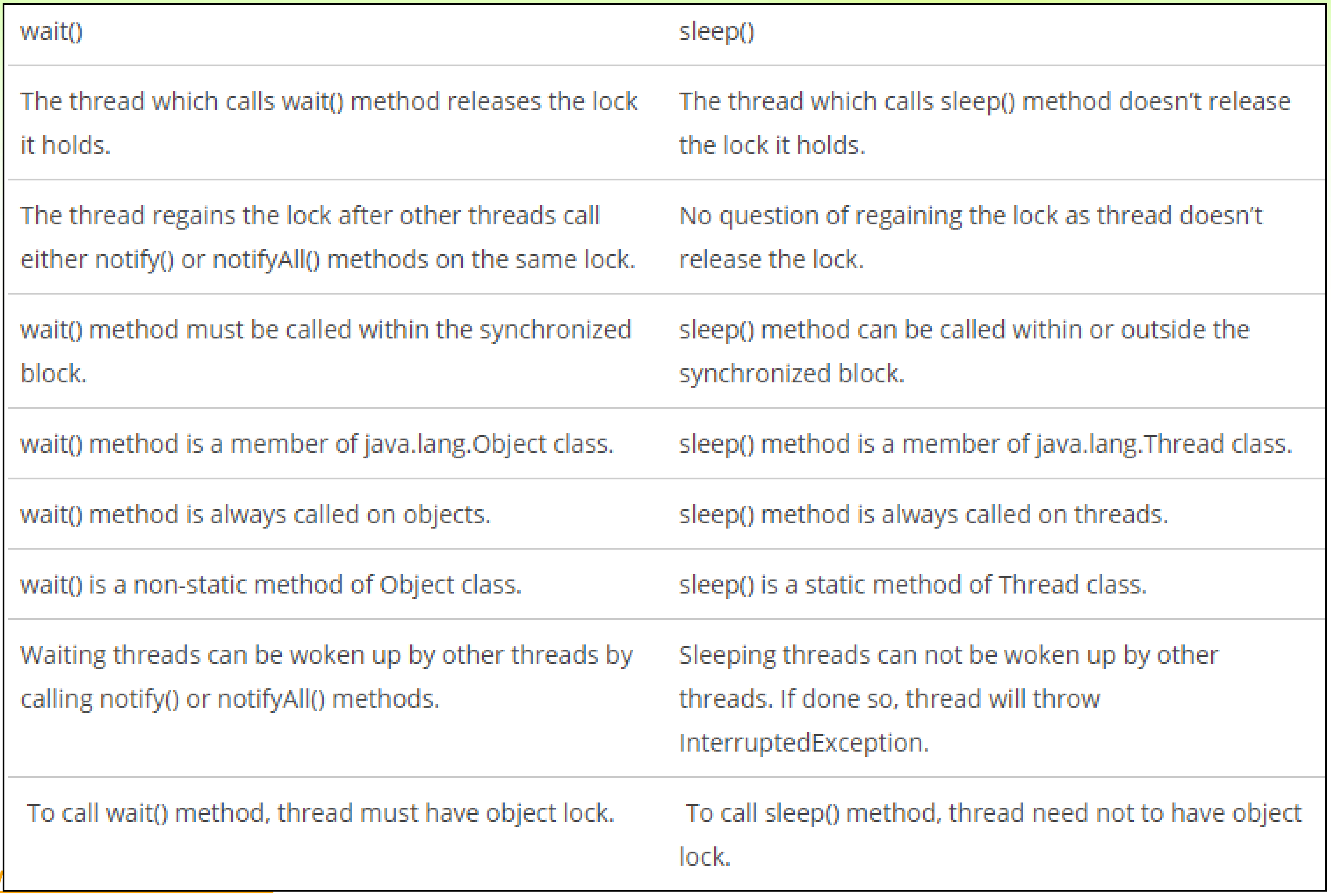
WAIT() VS. SLEEP()

❖wait() and sleep() methods in java, are used to pause the execution of a particular thread in a multi threaded environment.

❖Whenever a thread calls wait() method, it releases the lock or monitor it holds and whenever a thread calls sleep() method, it doesn’t release the lock or monitor it holds.

❖This is the main difference between sleep() and wait() methods.

wait() vs. sleep()



wait()	sleep()
The thread which calls wait() method releases the lock it holds.	The thread which calls sleep() method doesn't release the lock it holds.
The thread regains the lock after other threads call either notify() or notifyAll() methods on the same lock.	No question of regaining the lock as thread doesn't release the lock.
wait() method must be called within the synchronized block.	sleep() method can be called within or outside the synchronized block.
wait() method is a member of java.lang.Object class.	sleep() method is a member of java.lang.Thread class.
wait() method is always called on objects.	sleep() method is always called on threads.
wait() is a non-static method of Object class.	sleep() is a static method of Thread class.
Waiting threads can be woken up by other threads by calling notify() or notifyAll() methods.	Sleeping threads can not be woken up by other threads. If done so, thread will throw InterruptedException.
To call wait() method, thread must have object lock.	To call sleep() method, thread need not to have object lock.

http://www

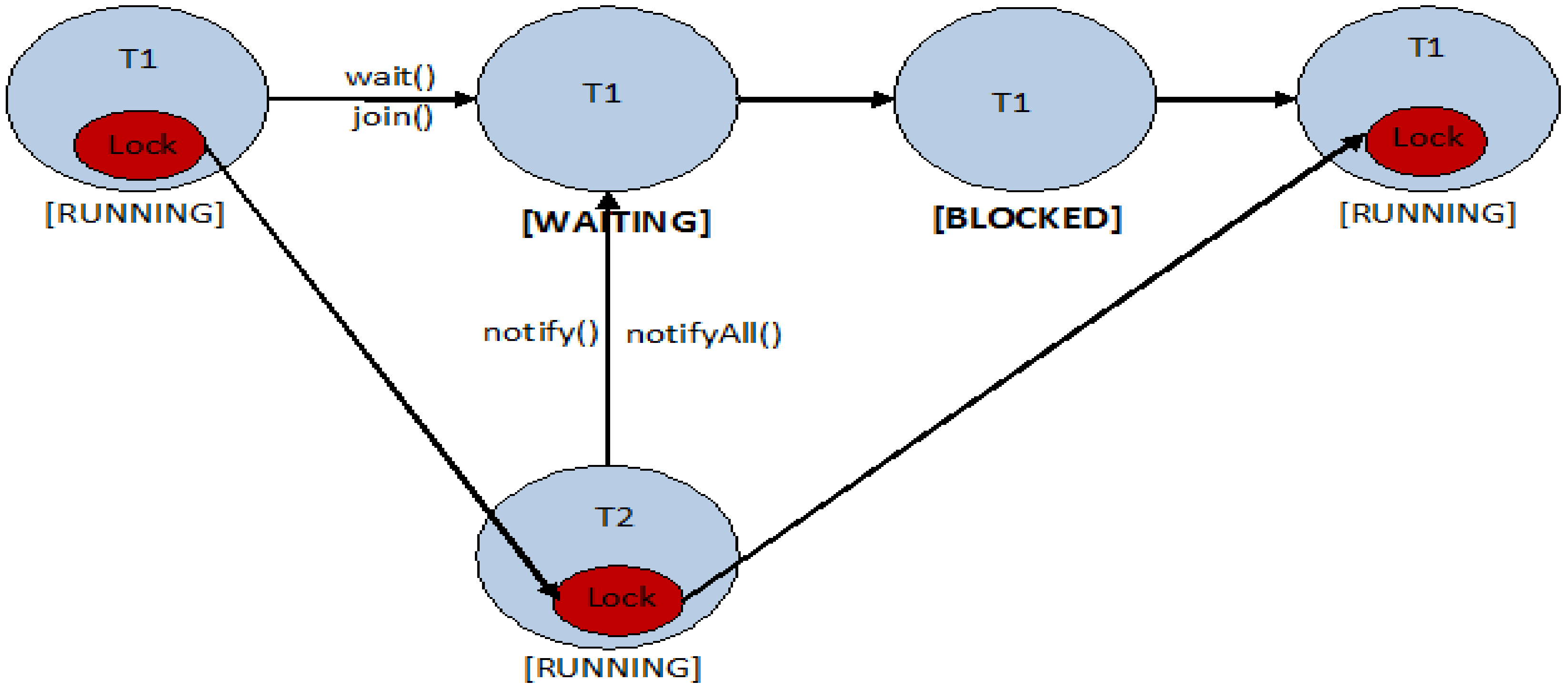
Blocked vs. Waiting

❖A thread enters into WAITING state when it calls wait() or join() method on an object. Before entering into WAITING state, thread releases the lock of the object it holds. It will remain in WAITING state until any other thread calls either notify() or notifyAll() on the same object.

❖Once the other thread calls notify() or notifyAll() on the same object, one or all the threads which are WAITING for lock of that object will be notified. All the notified threads will not get the object lock immediately. They will get the object lock on a priority basis once the current thread releases the lock. Until that they will be in BLOCKED state.

❖In simple terms, a thread will be in WAITING state if it is waiting for notification from other threads. A thread will be in BLOCKED state if it is waiting for other thread to release the lock it wants.

BLOCKED Vs WAITING



WAITING : The thread will be in this state when it calls *wait()* or *join()* method.

BLOCKED : The thread will be in this state when it is notified by other thread but has not got the object lock yet.

http

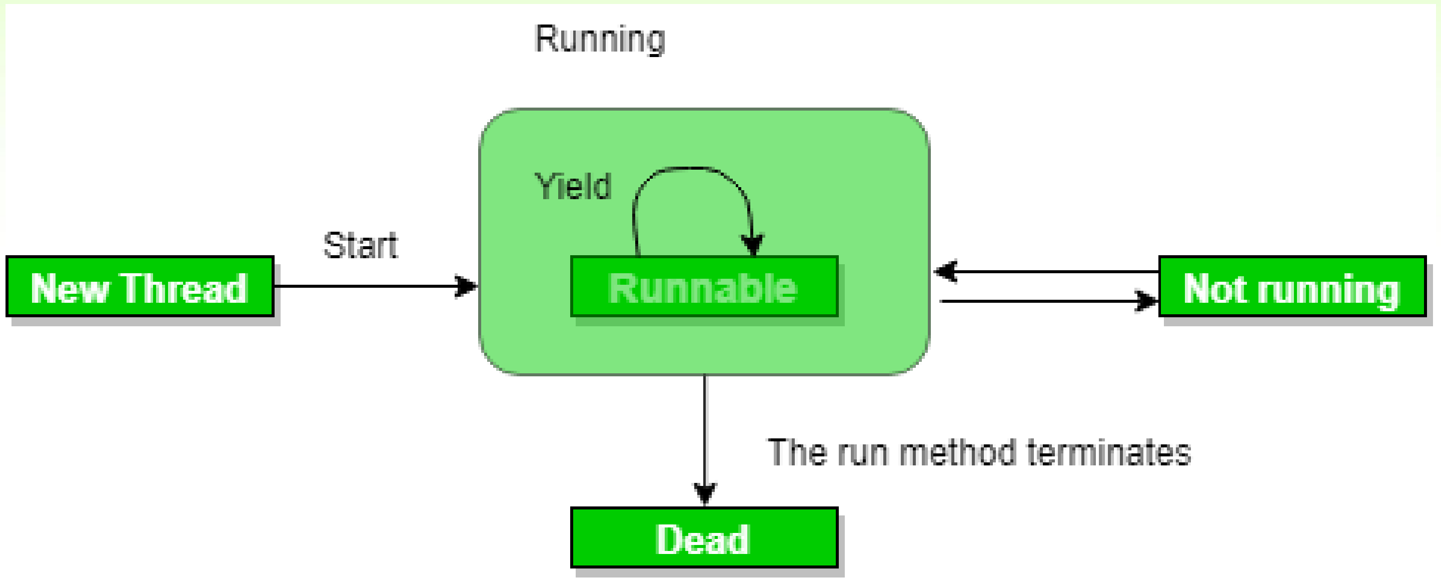
YIELD(), SLEEP() AND JOIN() METHODS

❖**yield():**

❖Suppose there are three threads t1, t2, and t3. Thread t1 gets the processor and starts its execution and thread t2 and t3 are in Ready/Runnable state. Completion time for thread t1 is 5 hour and completion time for t2 is 5 minutes. Since t1 will complete its execution after 5 hours, t2 has to wait for 5 hours to just finish 5 minutes job. In such scenarios where one thread is taking too much time to complete its execution, we need a way to prevent execution of a thread in between if something important is pending. yeild() helps us in doing so.

❖yield() basically means that the thread is not doing anything particularly important and if any other threads or processes need to be run, they should run. Otherwise, the current thread will continue to run.

Yield(), sleep() and join() methods



Yield(), sleep() and join() Methods

❖Use of yield method:

❖Whenever a thread calls java.lang.Thread.yield method, it gives hint to the thread scheduler that it is ready to pause its execution. Thread scheduler is free to ignore this hint.

❖If any thread executes yield method , thread scheduler checks if there is any thread with same or high priority than this thread. If processor finds any thread with higher or same priority then it will move the current thread to Ready/Runnable state and give processor to other thread and if not – current thread will keep executing.

Yield(), sleep() and join() methods

❖Use of yield method:

❖Whenever a thread calls java.lang.Thread.yield method, it gives hint to the thread scheduler that it is ready to pause its execution. Thread scheduler is free to ignore this hint.

❖If any thread executes yield method , thread scheduler checks if there is any thread with same or high priority than this thread. If processor finds any thread with higher or same priority then it will move the current thread to Ready/Runnable state and give processor to other thread and if not – current thread will keep executing.

Yield(), sleep() and join() methods

❖Once a thread has executed yield method and there are many threads with same priority is waiting for processor, then we can't specify which thread will get execution chance first.

❖The thread which executes the yield method will enter in the Runnable state from Running state.

❖Once a thread pauses its execution, we can't specify when it will get chance again it depends on thread scheduler.

❖Underlying platform must provide support for preemptive scheduling if we are using yield method.

Yield(), Sleep() and Join() Methods

❖**sleep():**

❖This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

YIELD( ) VS SLEEP( )

❖**yield:()** indicates that the thread is not doing anything particularly important and if any other threads or processes need to be run, they can. Otherwise, **the current thread will continue to run.**

❖**sleep():** causes the thread to definitely stop executing for a given amount of time; if no other thread or process needs to be run**, the CPU will be idle** (and probably enter a power saving mode).

Yield(), Sleep() and Join() Methods

❖**join():**

❖The join() method of a Thread instance is used to join the start of a thread's execution to end of other thread's execution such that a thread does not start running until another thread ends. If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing.

❖The join() method waits at most this much milliseconds for this thread to die.

❖If any executing thread t1 calls join() on t2 i.e; t2.join() immediately t1 will enter into waiting state until t2 completes its execution.

QUESTIONS

❖Differentiate between process and thread?

❖A Program in the execution is called the process whereas; A thread is a subset of the process

❖Processes are independent whereas threads are the subset of process.

❖Process have different address space in memory, while threads contain a shared address space.

❖Context switching can be faster between the threads as compared to context switching between the threads.

❖Inter-process communication is slower and expensive than inter-thread communication.

❖Any change in Parent process doesn't affect the child process whereas changes in parent thread can affect the child thread.

QUESTIONS

❖What do you understand by inter-thread communication?

❖The process of communication between synchronized threads is termed as inter-thread communication.

❖Inter-thread communication is used to avoid thread polling in Java.

❖The thread is paused running in its critical section, and another thread is allowed to enter (or lock) in the same critical section to be executed.

❖It can be obtained by wait(), notify(), and notifyAll() methods.

QUESTIONS

❖What is the purpose of wait() method in Java?

❖The wait() method is provided by the Object class in Java. This method is used for inter-thread communication in Java. The java.lang.Object.wait() is used to pause the current thread, and wait until another thread does not call the notify() or notifyAll() method. Its syntax is given below.

❖public final void wait()

QUESTIONS

❖Why must wait() method be called from the synchronized block?

❖We must call the wait method otherwise it will throw java.lang.IllegalMonitorStateException exception.

❖Moreover, we need wait() method for inter-thread communication with notify() and notifyAll(). Therefore It must be present in the synchronized block for the proper and correct communication.

QUESTIONS

❖Differentiate between the Thread class and Runnable interface for creating a Thread?

❖The Thread can be created by using two ways.

❖By extending the Thread class
❖By implementing the Thread class
❖However, the primary differences between both the ways are given below:

QUESTIONS

❖By extending the Thread class, we cannot extend any other class, as Java does not allow multiple inheritances while implementing the Runnable interface; we can also extend other base class(if required).

❖By extending the Thread class, each of thread creates the unique object and associates with it while implementing the Runnable interface; multiple threads share the same object

❖Thread class provides various inbuilt methods such as getPriority(), isAlive and many more while the Runnable interface provides a single method, i.e., run().

QUESTIONS

❖ What does 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. Join method is overloaded in Thread class in the following ways.

❖public void join()throws InterruptedException
❖public void join(long milliseconds)throws InterruptedException

QUESTIONS

❖Can we make the user thread as daemon thread if the thread is started?

❖No, if you do so, it will throw IllegalThreadStateException. Therefore, we can only create a daemon thread before starting the thread.

QUESTIONS

❖What will happen if two thread of the same priority are called to be processed simultaneously?

❖a) Anyone will be executed first lexographically

❖b) Both of them will be executed simultaneously

❖c) None of them will be executed

❖d) It is dependent on the operating system

❖Answer: d

❖Explanation: In cases where two or more thread with same priority are competing for CPU cycles, different operating system handle this situation differently. Some execute them in time sliced manner some depending on the thread they call.

QUESTIONS

- ❖Which of these statements is incorrect?
- ❖a) By multithreading CPU idle time is minimized, and we can take maximum use of it
- ❖b) By multitasking CPU idle time is minimized, and we can take maximum use of it
- ❖c) Two thread in Java can have the same priority
- ❖d) A thread can exist only in two states, running and blocked

- ❖Answer: d
- ❖Explanation: Thread exist in several states, a thread can be running, suspended, blocked, terminated & ready to run.

QUESTIONS

❖What is the output of this program?

```
❖    class multithreaded_programing
❖    {
❖        public static void main(String args[])
❖        {
❖            Thread t = Thread.currentThread();
❖            System.out.println(t);
❖        }
❖    }
```

❖a) Thread[5,main].

❖b) Thread[main,5].

❖c) Thread[main,0].

❖d) Thread[main,5,main].

❖Answer: d

QUESTIONS

❖What is the name of the thread in output of this program?

```
class multithreaded_programing
{
    public static void main(String args[])
    {
        Thread t = Thread.currentThread();
        System.out.println(t);
    }
}
```

❖a) main

❖b) Thread

❖c) System

❖d) None of the mentioned

QUESTIONS

- Answer: a
- Explanation: The output of program is Thread[main,5,main], Since we have not explicitly named the thread they are named by the group to they belong i:e main method. Hence they are named ‘main’.

QUESTIONS

❖1. public class Threads2 implements Runnable {

❖2.

❖3. public void run() {

❖4. System.out.println("run.");

❖5. throw new RuntimeException("Problem");

❖6. }

❖7. public static void main(String[] args) {

❖8. Thread t = new Thread(new Threads2());

❖9. t.start();

❖10. System.out.println("End of method.");

❖11. }

❖12. }

QUESTIONS

- ❖Which two can be results? (Choose two.)
- ❖A. java.lang.RuntimeException: Problem
- ❖B. run.
- ❖java.lang.RuntimeException: Problem
- ❖C. End of method.
- ❖java.lang.RuntimeException: Problem
- ❖D. End of method.
- ❖run.
- ❖java.lang.RuntimeException: Problem
- ❖E. run.
- ❖java.lang.RuntimeException: Problem
- ❖End of method.

- ❖Answer: D, E

LAB 7

❖Q1. Write a program to print numbers from 1 to 10 with a delay of 1 second between each print.

❖Q2. Write a program to create 3 threads with different priorities and call start() method for each thread and print thread name along with priority.

❖Q3. Write a program in which a thread is created and print “Thread is running”