BCA – 401: Java Programming

Rahul Kumar Singh

In today's Class we have discussed on Multithreading in Java.

Using thread methods in Java:-

sleep() method in Java:-

The Java Thread class provides the two variant of the sleep() method. First one accepts only one arguments, whereas the other variant accepts two arguments. The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is known as the sleeping time of the thread. After the sleeping time is over, the thread starts its execution from where it has left.

Syntax of the sleep() method:-

public static void sleep(long mls) throws InterruptedException

public static void sleep(long mls, int n) throws InterruptedException

The method sleep() with the one parameter is the native method, and the implementation of the native method is accomplished in another programming language.

The other methods having the two parameters are not the native method. That is, its implementation is accomplished

in Java.

Parameters:-

The following are the parameters used in the sleep() method.

mls: milliseconds Duration of time to make a thread sleep

n: nanoseconds Additional duration of time to make thread sleep but restricted to 999999. The range of n is from 0 to 999999.

The method does not return anything.

Important Points to Remember About the Sleep() Method:-

- Whenever the Thread.sleep() methods execute, it always halts the execution of the current thread.
- Whenever another thread does interruption while the current thread is already in the sleep mode, then the InterruptedException is thrown.
- If the system that is executing the threads is busy, then the actual sleeping time of the thread is generally more as compared to the time passed in arguments. However, if the system executing the sleep() method has less load, then the actual sleeping time of the thread is almost equal to the time passed in the argument.

Example of the sleep() method in Java.

```
class TestSleepMethod1 extends Thread
{
 public void run()
{
  for(int i=1;i<5;i++)
{
  // the thread will sleep for the 500 milliseconds
    try
{
Thread.sleep(500);
}
catch(InterruptedException e)
{
System.out.println(e);
}
    System.out.println(i);
  }
 }
 public static void main(String args[])
```

```
{
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();
  t1.start();
  t2.start();
 }
}
```

Output:-

```
1
1
2
2
3
3
4
4
```

As you know well that at a time only one thread is executed. If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

Java join() method:-

The join() method in Java is provided by the java.lang.Thread class that permits one thread to wait until the other thread to finish its execution.

Description of The Overloaded join() Method:-

join(): When the join() method is invoked, the current thread stops its execution and the thread goes into the wait state. The current thread remains in the wait state until the thread on which the join() method is invoked has achieved its dead state. If interruption of the thread occurs, then it throws the InterruptedException.

Syntax:

```
public final void join() throws InterruptedException
```

Examples of the join() Method:-

```
class TestJoinMethod1 extends Thread
{
 public void run()
{
  for(int i=1;i<=5;i++)
{
   try
```

```
{
    Thread.sleep(500);
   }
catch(Exception e)
{
System.out.println(e);
}
  System.out.println(i);
  }
 }
public static void main(String args[])
{
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try
{
  t1.join();
 }
```

```
catch(Exception e){System.out.println(e);
}
 t2.start();
 t3.start();
 }
}
```

Output:-

```
1
2
3
4
5
1
1
2
2
3
3
4
4
```

5

5

We can see in the above example, when t1 completes its task then t2 and t3 starts executing.

join(long miliseconds) Method Example:-

```
class TestJoinMethod2 extends Thread
{
 public void run()
{
  for(int i=1;i<=5;i++)
{
   try
{
    Thread.sleep(500);
   }
catch(Exception e)
{
System.out.println(e);
}
  System.out.println(i);
```

```
  }
 }
public static void main(String args[])
{
 TestJoinMethod2 t1=new TestJoinMethod2();
 TestJoinMethod2 t2=new TestJoinMethod2();
 TestJoinMethod2 t3=new TestJoinMethod2();
 t1.start();
 try
{
  t1.join(1500);
 }
catch(Exception e)
{
System.out.println(e);
}
 t2.start();
 t3.start();
 }
}
```

Output:-

1

2

3

1

4

1

2

5

2

3

3

4

4

5

5

In the above example,

when t1 completes its task for 1500 milliseconds(3 times), then t2 and t3 start executing.

Java Thread Exceptions:-

Exception:-

An exception is an event that occurs during the execution of a program that disrupts the normal flow of instructions during the execution of a program. Exceptions are objects that represent errors that may occur in a Java program.

Java run system will throw IllegalThreadStateException whenever we attempt to invoke a method that a thread cannot handle in the given state. For example, a sleeping thread cannot deal with the resume() method because a sleeping thread cannot receive any instructions. The same is true with the suspend() method when it is used on a blocked (Not Runnable) thread.

Whenever we call a thread method that is likely to throw an exception, we have to supply an appropriate exception handler to catch it. The catch statement may take one of the following forms.

```
catch (ThreadDeath e)
{.....................
.....................     // killed the thread
}
catch (InterruptedException e)
{.....................
```

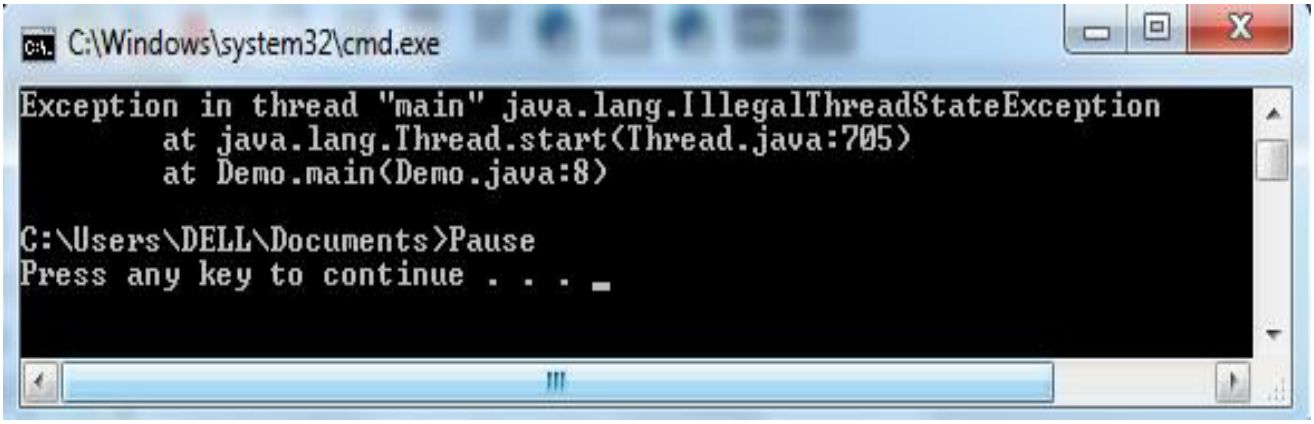
```
.....      // cannot handle it in the current state
}
catch (Illegal ArgumentException e)
{.....
.....      //illegal method argument
}
catch (Exception e)
{.....
.....      // Any other
}
```

Example:-

```
public class Demo extends Thread
{
  public static void main(String args[])
  {
     Demo d1 = new Demo();
     d1.start();
     d1.start();
  }
```

}

Output:-



Exceptions in Main Thread:-

In this section, we will see some common main thread exceptions that may occur in different scenarios. They are as follows:

1. **Exception in thread main java.lang.UnsupportedClassVersionError:** This exception occurs in a program when a java class is compiled from another JDK version and we are trying to run it from another java version.

The UnsupportedClassVersionError is present java.lang package.

2. **Exception in thread main java.lang.NoClassDefFoundError:** This exception occurs in two flavors. The first scenario is where we provide class full name with .class extension. The second scenario comes when Class is not found.

3. Exception in thread main java.lang.NoSuchMethodError: main: This exception occurs in a Java program when a class is trying to run without the main method declaration.

4. Exception in thread main java.lang.ArithmeticException: When any exception is thrown from main method, it prints the exception in the console.

Exception in sleep() method:-

When we call a sleep() method in a Java program, it must be enclosed in try block and followed by catch block.

This is because sleep() method throws an exception named InterruptedException that should be caught. If we fail to catch this exception, the program will not compile.

JVM (Java Runtime System) will throw an exception named IllegalThreadStateException whenever we attempt to call a method that a thread cannot handle in the given state.

For example, a thread that is in a sleeping state cannot deal with the resume() method because a sleeping thread cannot accept instructions. The same thing is true for the suspend() method when it is used on a blocked thread.

When we call a thread method that may throw an exception, we will have to use an appropriate exception handler to catch it.

Exception: It does often throws out exceptions as java language being involving the concept of multithreading.

IllegalArgumentException is thrown when the parametric value is negative as it is bounded as discussed between [0 — +999999].

InterrupteException is thrown when a thread is interrupted with an ongoing thread as discussed java supports the concepts of multithreading.

Example:-

```
// Java program to Use exceptions with thread

// Importing Classes/Files

import java.io.*;

class GFG extends Thread

{

    public void run()

    {

        System.out.println("Throwing in "

                           + "MyThread");

        throw new RuntimeException();

    }

}

public class Main

{
```

```
    public static void main(String[] args)
    {
        GFG t = new GFG();
        t.start();
        // try block to deal with exception
        try
        {
            Thread.sleep(2000);
        }
        // catch block to handle the exception
        catch (Exception x)
        {
            // Print command when exception encountered
            System.out.println("Exception" + x);
        }
     // Print command just to show program run successfully
        System.out.println("Completed");
    }
}
```

Output:-

```
Throwing in MyThread
Exception in thread "Thread-0" java.lang.RuntimeException
        at testapp.MyThread.run(Main.java:19)
Completed
```

Priority of a Thread (Thread Priority):-

Each thread has a priority. Priorities are represented by a number between 1 and 10. In most cases, the thread scheduler 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. Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

Setter & Getter Method of Thread Priority:-

Let's discuss the setter and getter method of the thread priority.

public final int getPriority(): The java.lang.Thread.getPriority() method returns the priority of the given thread.

public final void setPriority(int newPriority): The

java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

Three constants defined in Thread class:-

- public static int MIN_PRIORITY
- public static int NORM_PRIORITY
- 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.

Example of priority of a Thread:

```
import java.lang.*;
public class ThreadPriorityExample extends Thread
{
public void run()
{
System.out.println("Inside the run() method");
}
public static void main(String argvs[])
{
```

```
ThreadPriorityExample th1 = new ThreadPriorityExample();

ThreadPriorityExample th2 = new ThreadPriorityExample();

ThreadPriorityExample th3 = new ThreadPriorityExample();

// We did not mention the priority of the thread. Therefore, the priorities of the thread is 5, the default value

// 1st Thread Displaying the priority of the thread using the getPriority() method

System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 2nd Thread Display the priority of the thread

System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 3rd Thread Display the priority of the thread

System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// Setting priorities of above threads by passing integer arguments

th1.setPriority(6);

th2.setPriority(3);

th3.setPriority(9);

System.out.println("Priority of the thread th1 is : " + th1.getPriority());
```

```
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

System.out.println("Priority of the thread th3 is : " + th3.getPriority());

// Displaying name of the currently executing thread

System.out.println("Currently Executing The Thread : " + Thread.currentThread().getName());

System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());

// Priority of the main thread is 10 now

Thread.currentThread().setPriority(10);

System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());

}

}
```

Output:

```
Priority of the thread th1 is : 5

Priority of the thread th2 is : 5

Priority of the thread th2 is : 5

Priority of the thread th1 is : 6

Priority of the thread th2 is : 3

Priority of the thread th3 is : 9
```

Currently Executing The Thread : main

Priority of the main thread is : 5

Priority of the main thread is : 10

We know that a thread with high priority will get preference over lower priority threads when it comes to the execution of threads.

However, there can be other scenarios where two threads can have the same priority. All of the processing, in order to look after the threads, is done by the Java thread scheduler. Refer to the following example to comprehend what will happen if two threads have the same priority.

Example:-

```
import java.lang.*;

public class ThreadPriorityExample1 extends Thread

{

public void run()

{

System.out.println("Inside the run() method");

}

public static void main(String argvs[])

{
```

```
 Thread.currentThread().setPriority(7);
System.out.println("Priority of the main thread is : " + Thread.currentThread().getPriority());
ThreadPriorityExample1 th1 = new ThreadPriorityExample1();
// th1 thread is the child of the main thread, therefore, the th1 thread also gets the priority 7
// Displaying the priority of the current thread
System.out.println("Priority of the thread th1 is : " + th1.getPriority());
}
}
```

Output:-

```
Priority of the main thread is : 7
Priority of the thread th1 is : 7
```

Explanation: If there are two threads that have the same priority, then one can not predict which thread will get the chance to execute first. The execution then is dependent on the thread scheduler's algorithm (First Come First Serve, Round-Robin, etc.)

Synchronization in Java:-

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization?

The synchronization is mainly used to:-

- To prevent thread interference.
- To prevent consistency problem.

Types of Synchronization:-

There are two types of synchronization:-

- Process Synchronization.
- Thread Synchronization.

Here, we will discuss only thread synchronization.

Thread Synchronization:-

There are two types of thread synchronization mutual exclusive and inter-thread communication.

- Mutual Exclusive
- Cooperation (Inter-thread communication in java)

Mutual Exclusive:-

Mutual Exclusive helps keep threads from interfering with one another while sharing data.

Mutual Exclusive can be achieved by using the **Synchronized Method**.

Concept of Lock in Java:-

Synchronization is built around an internal entity known as the lock or monitor. Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.

From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Java Synchronized Method:-

If we declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Example:-

```
class Table
{
 synchronized void printTable(int n)
  {//synchronized method
   for(int i=1;i<=5;i++)
   {
     System.out.println(n*i);
     try
     {
      Thread.sleep(400);
     }
     catch(Exception e)
     {
        System.out.println(e);
        }
   }
 }
}
class MyThread1 extends Thread
```

```
{
Table t;
MyThread1(Table t)
{
this.t=t;
}
public void run()
{
t.printTable(5);
}
}
class MyThread2 extends Thread
{
Table t;
MyThread2(Table t)
{
this.t=t;
}
public void run()
{
```

```
t.printTable(100);
}
}
public class TestSynchronization2
{
public static void main(String args[])
{
Table obj = new Table(); //only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Output:-

```
  5
 10
 15
 20
 25
```

100

200

300

400

500

Java Block Synchronization:-

Synchronized block is used to lock an object for any shared resource.

Scope of synchronized block is smaller than the method.

A Java synchronized block doesn't allow more than one JVM, to provide access control to a shared resource.

The system performance may degrade because of the slower working of synchronized keyword.

Java synchronized block is more efficient than Java synchronized method.

Syntax:-

```
synchronized (object reference expression)
{
  //code block
}
```

Example:-

```
class Table
{
 void printTable(int n)
   {
     synchronized(this)
     {
      //synchronized block
     for(int i=1;i<=5;i++)
     {
      System.out.println(n*i);
      try
      {
       Thread.sleep(400);
      }
        catch(Exception e)
        {
            System.out.println(e);
         }
     }
```

```
  }
 }        //end of the method
}
class MyThread1 extends Thread
{
Table t;
MyThread1(Table t)
{
this.t=t;
}
public void run()
{
t.printTable(5);
}
}
class MyThread2 extends Thread
{
Table t;
MyThread2(Table t)
{
```

```
this.t=t;
}
public void run()
{
t.printTable(100);
}
}
public class TestSynchronizedBlock1
{
public static void main(String args[])
{
Table obj = new Table();   //only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Output:-

```
5
```

10

15

20

25

100

200

300

400

500

Inter-thread Communication in Java:-

Inter-thread communication or Co-operation is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a 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 is implemented by following methods of Object class:

wait()

notify()

notifyAll()

wait() method:-

The wait() method causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

public final void wait()throws InterruptedException :- It waits until object is notified.

public final void wait(long timeout)throws InterruptedException :- It waits for the specified amount of time.

notify() method:-

The notify() method wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation.

Syntax:-

public final void notify()

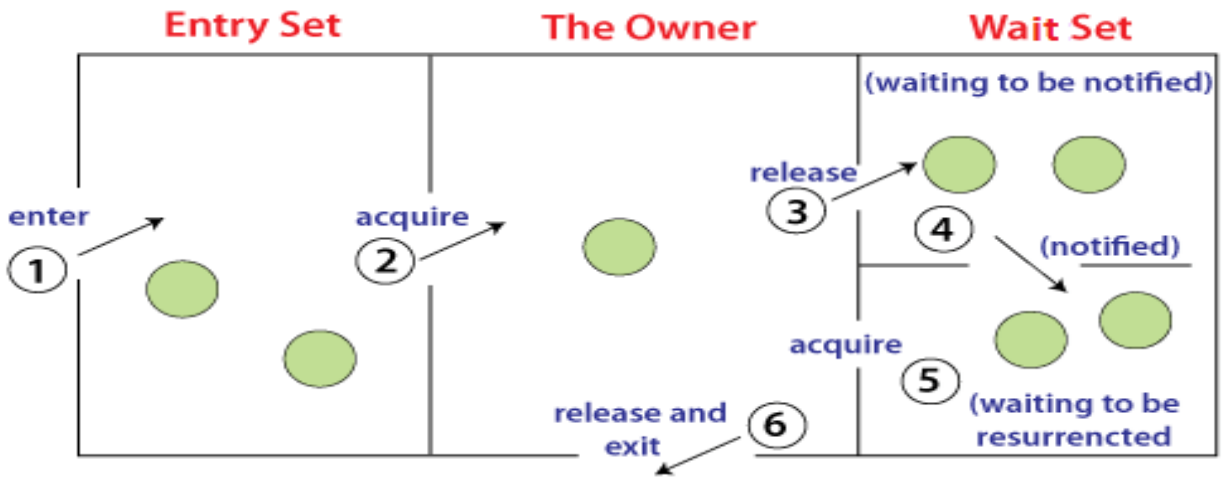
notifyAll() method:-

Wakes up all threads that are waiting on this object's monitor.

Syntax:

public final void notifyAll()

Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

1. Threads enter to acquire lock.
2. Lock is acquired by on thread.
3. Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
4. If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
5. Now thread is available to acquire lock.

6. After completion of the task, thread releases the lock and exits the monitor state of the object.

Why wait(), notify() and notifyAll() methods are defined in Object class not Thread class?

It is because they are related to lock and object has a lock.

Example of Inter Thread Communication:-

```
class Customer
{
int amount=10000;
synchronized void withdraw(int amount)
{
System.out.println("going to withdraw...");
if(this.amount<amount)
{
System.out.println("Less balance; waiting for deposit...");
try
{
wait();
}
catch(Exception e)
{
```

```
}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}
synchronized void deposit(int amount)
{
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}
class Test
{
public static void main(String args[])
{
final Customer c=new Customer();
new Thread()
{
```

```
public void run()
{
c.withdraw(15000);
}
}.start();
new Thread()
{
public void run()
{
c.deposit(10000);
}
}.start();
}
}
```

Output:

```
going to withdraw...
Less balance; waiting for deposit...
going to deposit...
deposit completed...
withdraw completed
```