Java Thread Class

The Java Thread class provides methods and constructors to create and perform operations on a thread. The Java thread class extends the Object class and implements the Runnable interface.

Java Thread Methods

These are the methods that are available in the Thread class:

1. public void start()

It starts the execution of the thread and then calls the ***run()*** on this Thread object.

Example:

```
1
2     {
3         public void run()
4         {
5             System.out.println("Thread is running...");
6         }
7         public static void main(String args[])
8         {
9             StartExp1 thread1=new StartExp1();
10            thread1.start();
11        }
    }
```

Output:

Thread is running…

2. public void run()

This thread is used to do an action for a thread. The run() method is instantiated if the thread was constructed using a separate Runnable object.

Example:

```
1     public class RunExp1 implements Runnable
2     {
3         public void run()
4         {
              System.out.println("Thread is running...");
```

```
    }
    public static void main(String args[])
    {
        RunExp1 r1=new RunExp1();
        Thread thread1 =new Thread(r1);
        thread1.start();
    }
}
```

Output:

Thread is running…

3. public static void sleep()

This blocks the currently running thread for the specified amount of time.

Example:

```
public class SleepExp1 extends Thread
{
    public void run()
    {
        for(int i=1;i<5;i++)
        {
            try
            {
                Thread.sleep(500);
            }catch(InterruptedException e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        SleepExp1 thread1=new SleepExp1();
        SleepExp1 thread2=new SleepExp1();
        thread1.start();
        thread2.start();
    }
}
```

Output:

1

1

2

2

3

3

4

4

4. public static Thread currentThread()

It returns a reference to the currently running thread.

Example:

```
1
2     public class CurrentThreadExp extends Thread
3     {
4         public void run()
5         {
6             System.out.println(Thread.currentThread().getName());
7         }
8         public static void main(String args[])
9         {
10            CurrentThreadExp thread1=new CurrentThreadExp();
11            CurrentThreadExp thread2=new CurrentThreadExp();
12            thread1.start();
13            thread2.start();
14        }
    }
```

Output:

Thread-0

Thread-1

5. public void join()

It causes the current thread to block until the second thread terminates or the specified amount of milliseconds passes.

Example:

```
1     public class JoinExample1 extends Thread
2     {
```

```
3         public void run()
4         {
5             for(int i=1; i<=4; i++)
6             {
7                 try
8                 {
9                     Thread.sleep(500);
10                }catch(Exception e){System.out.println(e);}
11                System.out.println(i);
12            }
13        }
14        public static void main(String args[])
15        {
16            JoinExample1 thread1 = new JoinExample1();
17            JoinExample1 thread2 = new JoinExample1();
18            JoinExample1 thread3 = new JoinExample1();
19            thread1.start();
20           try
21            {
22            thread1.join();
23            }catch(Exception e){System.out.println(e);}
24            thread2.start();
25            thread3.start();
26        }
27    }
```

Output:

1

2

3

4

1

1

2

2

3

3

4

4

6. public final int getPriority()

It is used to check the priority of the thread. When a thread is created, some priority is assigned to it. This priority is assigned either by the JVM or by the programmer explicitly while creating the thread.

Example:

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

```
public class JavaGetPriorityExp extends Thread
{
    public void run()
    {
        System.out.println("running thread name is:"+Thread.currentThread().getNam
    }
    public static void main(String args[])
    {
        JavaGetPriorityExp t1 = new JavaGetPriorityExp();
        JavaGetPriorityExp t2 = new JavaGetPriorityExp();
        System.out.println("t1 thread priority : " + t1.getPriority());
        System.out.println("t2 thread priority : " + t2.getPriority());
        t1.start();
        t2.start();
    }
}
```

Output:

t1 thread priority : 5

t2 thread priority : 5

running thread name is:Thread-0

running thread name is:Thread-1

7. public final void setPriority()

This method is used to change the priority of the thread. The priority of every thread is represented by the integer number from 1 to 10. The default priority of a thread is 5.

Example:

```
1
2     public class JavaSetPriorityExp1 extends Thread
3     {
4         public void run()
5         {
6             System.out.println("Priority of thread is: "+Thread.currentThread().getPri
7         }
8         public static void main(String args[])
9         {
10            JavaSetPriorityExp1 t1=new JavaSetPriorityExp1();
11            t1.setPriority(Thread.MAX_PRIORITY);
12            t1.start();
13        }
14    }
15
```

Output:

Priority of thread is: 10

8. public final String getName()

This method of thread class is used to return the name of the thread. We cannot override this method in our program, as this method is final.

Example:

```
1
2
3     public class GetNameExample extends Thread
4     {
5         public void run()
6         {
7             System.out.println("Thread is running...");
8         }
9         public static void main(String args[])
10        {
11            // creating two threads
12            GetNameExample thread1=new GetNameExample();
13            GetNameExample thread2=new GetNameExample();
14            System.out.println("Name of thread1: "+ thread1.getName());
15            System.out.println("Name of thread2: "+thread2.getName());
16            thread1.start();
17            thread2.start();
18        }
19    }
20
```

Output:

Name of thread1: Thread-0

Name of thread2: Thread-1

Thread is running…

Thread is running…

9. public final void setName()

This method changes the name of the thread.

Example:

```
1
2
3     public class SetNameExample extends Thread
4     {
5         public void run()
6         {
7             System.out.println("running...");
8         }
9         public static void main(String args[])
10        {
11            SetNameExample thread1=new SetNameExample();
12            SetNameExample thread2=new SetNameExample();
13            thread1.start();
14            thread2.start();
15            thread1.setName("Kadamb Sachdeva");
16            thread2.setName("Great learning");
17            System.out.println("After changing name of thread1: "+thread1.getName());
18            System.out.println("After changing name of thread2: "+thread2.getName());
        }
    }
```

Output:

After changing name of thread1: Kadamb Sachdeva

After changing name of thread2: Great Learning

running…

running…

10. public long getId()

It returns the identifier of the thread. The thread ID is a number generated when the thread was created. This ID cannot be changed during its lifetime. But when the thread is terminated, the ID can be reused.

Example:

```
1
2     public class GetIdExample extends Thread
3     {
4         public void run()
5         {
6             System.out.println("running...");
7         }
8         public static void main(String args[])
9         {
10            GetIdExample thread1=new GetIdExample();
11            System.out.println("Name of thread1: "+thread1.getName());
12            System.out.println("Id of thread1: "+thread1.getId());
13            thread1.start();
14        }
    }
```

Output:

Name of thread1: Thread-0

Id of thread1: 21

running…

11. public final boolean isAlive()

This method checks if the thread is alive. A thread is in the alive state if the start() method of thread class has been called and the thread has not yet died.

Example:

```
1
2     public class JavaIsAliveExp extends Thread
3     {
4         public void run()
5         {
6             try
7             {
8                 Thread.sleep(300);
9                 System.out.println("is run() method isAlive "+Thread.currentThread().i
10            }
11            catch (InterruptedException ie) {
12            }
13        }
14        public static void main(String[] args)
15        {
16            JavaIsAliveExp thread1 = new JavaIsAliveExp();
17            System.out.println("before starting thread isAlive: "+thread1.isAlive());
18            thread1.start();
19            System.out.println("after starting thread isAlive: "+thread1.isAlive());
20        }
21    }
```

Output:

before starting thread isAlive: false

after starting thread isAlive: true

is run() method isAlive true

12. public static void yield()

This method pauses the execution of the current thread to execute other threads temporarily.

Example:

```
public class JavaYieldExp extends Thread
{
    public void run()
    {
        for (int i=0; i<3 ; i++)
            System.out.println(Thread.currentThread().getName() + " in control");
    }
    public static void main(String[]args)
    {
        JavaYieldExp thread1 = new JavaYieldExp();
        JavaYieldExp thread2 = new JavaYieldExp();
        thread1.start();
        thread2.start();
        for (int i=0; i<3; i++)
        {
            thread1.yield();
            System.out.println(Thread.currentThread().getName() + " in control");
        }
    }
}
```

Output:

main in control

main in control

main in control

Thread-0 in control

Thread-0 in control

Thread-0 in control

Thread-1 in control

Thread-1 in control

Thread-1 in control

13. public final void suspend()

This method is used to suspend the currently running thread temporarily. Using the resume() method, you can resume the suspended thread.

Example:

```
1
2
3     public class JavaSuspendExp extends Thread
4     {
5         public void run()
6         {
7             for(int i=1; i<5; i++)
8             {
9                 try
10                {
11                    sleep(500);
12                    System.out.println(Thread.currentThread().getName());
13                }catch(InterruptedException e){System.out.println(e);}
14                System.out.println(i);
15            }
16        }
17        public static void main(String args[])
18        {
19            JavaSuspendExp thread1=new JavaSuspendExp ();
20            JavaSuspendExp thread2=new JavaSuspendExp ();
21            JavaSuspendExp thread3=new JavaSuspendExp ();
22            thread1.start();
23            thread2.start();
24            thread2.suspend();
25            thread3.start();
        }
    }
```

Output:

Thread-0

1

Thread-2

1

Thread-0

2

Thread-2

2

Thread-0

3

Thread-2

3

Thread-0

4

Thread-2

4

14. public final void resume()

This method is used to resume the suspended thread. It is only used with the suspend() method.

Example:

```
1     public class JavaResumeExp extends Thread
2     {
3         public void run()
4         {
5             for(int i=1; i<5; i++)
6             {
                  try
```

```
                {
                     sleep(500);
                     System.out.println(Thread.currentThread().getName());
                }catch(InterruptedException e){System.out.println(e);}
                System.out.println(i);
            }
        }
        public static void main(String args[])
        {
            JavaResumeExp thread1=new JavaResumeExp ();
            JavaResumeExp thread2=new JavaResumeExp ();
            JavaResumeExp thread3=new JavaResumeExp ();
            thread1.start();
            thread2.start();
            thread2.suspend();
            thread3.start();
            thread2.resume();
        }
    }
```

Output:

Thread-0

1

Thread-2

1

Thread-1

1

Thread-0

2

Thread-2

2

Thread-1

2

Thread-0

3

Thread-2

3

Thread-1

3

Thread-0

4

Thread-2

4

Thread-1

4

15. public final void stop()

As the name suggests, this method is used to stop the currently running thread. Remember, once the thread execution is stopped, it cannot be restarted.

Example:

```
1   public class JavaStopExp extends Thread
2   {
3       public void run()
4       {
5           for(int i=1; i<5; i++)
6           {
7               try
8               {
9                   sleep(500);
10                  System.out.println(Thread.currentThread().getName());
11              }catch(InterruptedException e){System.out.println(e);}
12              System.out.println(i);
13          }
        }
        public static void main(String args[])
        {
```

```
            JavaStopExp thread1=new JavaStopExp ();
            JavaStopExp thread2=new JavaStopExp ();
            JavaStopExp thread3=new JavaStopExp ();
            thread1.start();
            thread2.start();
            thread3.stop();
            System.out.println("Thread thread3 is stopped");
        }
    }
```

Output:

16. public void destroy()

This thread method destroys the thread group as well as its subgroups.

Example:

```
public class JavaDestroyExp extends Thread
{
    JavaDestroyExp(String threadname, ThreadGroup tg)
    {
        super(tg, threadname);
        start();
    }
    public void run()
    {
        for (int i = 0; i < 2; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex) {
                System.out.println("Exception encounterted");}
        }
        System.out.println(Thread.currentThread().getName() +
              " finished executing");
    }
    public static void main(String arg[]) throws InterruptedException, SecurityExcep
    {
        ThreadGroup g1 = new ThreadGroup("Parent thread");
        ThreadGroup g2 = new ThreadGroup(g1, "child thread");
        JavaDestroyExp thread1 = new JavaDestroyExp("Thread-1", g1);
        JavaDestroyExp thread2 = new JavaDestroyExp("Thread-2", g1);
        thread1.join();
        thread2.join();
        g2.destroy();
        System.out.println(g2.getName() + " destroyed");
        g1.destroy();
        System.out.println(g1.getName() + " destroyed");
    }
}
```

Output:

Thread-1 finished executing

Thread-2 finished executing

child thread destroyed

Parent thread destroyed

17. public final boolean isDaemon()

This thread method will check if the thread is a daemon thread or not. If it is a daemon thread, then it will return true else, it will return false.

For those who don't know about a daemon thread, a daemon thread is a thread that will not stop the Java Virtual Machine (JVM) from exiting when the program is ended, but the thread is still running.

Example:

```
public class JavaIsDaemonExp extends Thread
{
    public void run()
    {
        //checking for daemon thread
        if(Thread.currentThread().isDaemon())
        {
            System.out.println("daemon thread work");
        }
        else
        {
            System.out.println("user thread work");
        }
    }
    public static void main(String[] args)
    {
        JavaIsDaemonExp thread1=new JavaIsDaemonExp();
        JavaIsDaemonExp thread2=new JavaIsDaemonExp();
        JavaIsDaemonExp thread3=new JavaIsDaemonExp();
        thread1.setDaemon(true);
        thread1.start();
        thread2.start();
        thread3.start();
    }
```

```
}
```

Output:

daemon thread work

user thread work

user thread work

18. public final void setDaemon(boolean on)

This method of a thread is used to identify or mark the thread either daemon or a user thread. The JVM automatically terminates this thread when all the user threads die.

This thread method must run before the start of the execution of the thread.

Example:

```

public class JavaSetDaemonExp1 extends Thread
{
    public void run()
    {
        if(Thread.currentThread().isDaemon())
        {
            System.out.println("daemon thread work");
        }
        else
        {
            System.out.println("user thread work");
        }
    }
    public static void main(String[] args)
    {
        JavaSetDaemonExp1 thread1=new JavaSetDaemonExp1();
        JavaSetDaemonExp1 thread2=new JavaSetDaemonExp1();
        JavaSetDaemonExp1 thread3=new JavaSetDaemonExp1();
        thread1.setDaemon(true);
        thread1.start();
        thread2.setDaemon(true);
        thread2.start();
        thread3.start();
    }
}

```

25

Output:

daemon thread work

daemon thread work

user thread work

19. public void interrupt()

This method of a thread is used to interrupt the currently executing thread. This method can only be called when the thread is in sleeping or waiting state.

But if the thread is not in the sleeping or waiting state, then the interrupt() method will not interrupt the thread but will set the interrupt flag to true.

Example:

```
1
2
3     public class JavaInterruptExp1 extends Thread
4     {
5         public void run()
6         {
7             try
8             {
9                 Thread.sleep(1000);
10                System.out.println("javatpoint");
11            }catch(InterruptedException e){
12                throw new RuntimeException("Thread interrupted..."+e);
13
14            }
15        }
16        public static void main(String args[])
17        {
18            JavaInterruptExp1 thread1=new JavaInterruptExp1();
19            thread1.start();
20            try
21            {
22                thread1.interrupt();
23            }catch(Exception e){System.out.println("Exception handled "+e);}
        }
    }
```

Output:

Exception in thread “Thread-0” java.lang.RuntimeException: Thread interrupted…java.lang.InterruptedException: sleep interrupted at JavaInterruptExp1.run(JavaInterruptExp1.java:10)

20. public boolean isInterrupted()

This thread method is used to test whether the thread is interrupted or not. It will return the value of the internal flag as true or false, i.e. if the thread is interrupted, it will return true else, it will return false.

Example:

```
1
2
3   public class JavaIsInterruptedExp extends Thread
4   {
5       public void run()
6       {
7           for(int i=1;i<=3;i++)
8           {
9               System.out.println("doing task....: "+i);
10          }
11      }
12      public static void main(String args[])throws InterruptedException
13      {
14          JavaIsInterruptedExp thread1=new JavaIsInterruptedExp();
15          JavaIsInterruptedExp thread2=new JavaIsInterruptedExp();
16          thread1.start();
17          thread2.start();
18          System.out.println("is thread interrupted..: "+thread1.isInterrupted());
19          System.out.println("is thread interrupted..: "+thread2.isInterrupted());
20          thread1.interrupt();
21          System.out.println("is thread interrupted..: " +thread1.isInterrupted());
22          System.out.println("is thread interrupted..: "+thread2.isInterrupted());
    }
}
```

Output:

is thread interrupted..: false

is thread interrupted..: false

is thread interrupted..: true

is thread interrupted..: false

doing task…..: 1

doing task…..: 2

doing task…..: 3

doing task….: 1

doing task….: 2

doing task….: 3

21. public static boolean interrupted()

This thread method is used to check if the current thread is interrupted or not. If this threading method is to be called twice in succession, then the second call will return as false.

If the interrupt status of the thread is true, then this thread method will set it to false.

Example:

```
1
2
3     public class JavaInterruptedExp extends Thread
4     {
5         public void run()
6         {
7             for(int i=1;i<=3;i++)
8             {
9                 System.out.println("doing task....: "+i);
10            }
11        }
12        public static void main(String args[])throws InterruptedException
13        {
14            JavaInterruptedExp thread1=new JavaInterruptedExp();
15            JavaInterruptedExp thread2=new JavaInterruptedExp();
16            thread1.start();
17            thread2.start();
18            System.out.println("is thread thread1 interrupted..:"+thread1.interrupted(
19            thread1.interrupt();
20            System.out.println("is thread thread1 interrupted..:"+thread1.interrupted(
21            System.out.println("is thread thread2 interrupted..:"+thread2.interrupted(
        }
    }
```

Output:

is thread thread1 interrupted..: false

is thread thread1 interrupted..: false

is thread thread2 interrupted..: false

doing task….: 1

doing task….: 2

doing task….: 3

doing task….: 1

doing task….: 2

doing task….: 3

22. public static int activeCount()

This method of the thread is used to return the no. of active threads in the currently executing thread's thread group.

The number returned by this threading method is only an estimate number as the number of threads dynamically changes while this method traverses internal data structures.

Example:

```
1
2
3     public class JavaActiveCountExp extends Thread
4     {
5         JavaActiveCountExp(String threadname, ThreadGroup tg)
6         {
7             super(tg, threadname);
8             start();
9         }
10        public void run()
11        {
12           System.out.println("running thread name is:"
13    +Thread.currentThread().getName());
14        }
15        public static void main(String arg[])
16        {
17            ThreadGroup g1 = new ThreadGroup("parent thread group");
18              JavaActiveCountExp thread1 = new JavaActiveCountExp("Thread-1", g1);
19            JavaActiveCountExp thread2 = new JavaActiveCountExp("Thread-2", g1);
20              System.out.println("number of active thread: "+ g1.activeCount());
21        }
22    }
```

Output:

number of active thread: 2

running thread name is: Thread-1

running thread name is: Thread-2

23. public final void checkAccess()

This thread method identifies if the current thread has permission to modify the thread.

Example:

```
1
2
3     public class JavaCheckAccessExp extends Thread
4     {
5         public void run()
6         {
7             System.out.println(Thread.currentThread().getName()+" finished executing")
8         }
9         public static void main(String arg[]) throws InterruptedException, SecurityExc
10        {
11            JavaCheckAccessExp thread1 = new JavaCheckAccessExp();
12            JavaCheckAccessExp thread2 = new JavaCheckAccessExp();
13            thread1.start();
14            thread2.start();
15            thread1.checkAccess();
16            System.out.println(t1.getName() + " has access");
17            thread2.checkAccess();
18            System.out.println(t2.getName() + " has access");
        }
    }
```

Output:

Thread-0 has access

Thread-1 has access

Thread-0 finished executing

Thread-1 finished executing

24. public static boolean holdsLock(Object obj)

This thread method checks if the currently executing thread holds the monitor lock on the specified object. If it does, then this threading method will return true.

Example:

```
1     public class JavaHoldLockExp implements Runnable
      {
```

```
2         public void run()
3         {
4             System.out.println("Currently executing thread is: " + Thread.currentThread
5             System.out.println("Does thread holds lock? " + Thread.holdsLock(this));
6             synchronized (this)
7             {
8                 System.out.println("Does thread holds lock? " + Thread.holdsLock(this))
9             }
10        }
11        public static void main(String[] args)
12        {
13            JavaHoldLockExp g1 = new JavaHoldLockExp();
14            Thread thread1 = new Thread(g1);
15            thread1.start();
16        }
17    }
18
```

Output:

Currently executing thread is: Thread-0

Does thread holds lock? false

Does thread holds lock? true

There are various thread methods that are used for different tasks and purposes. Those thread methods are as follows:

- public static void dumpStack()
- public StackTraceElement[] getStackTrace()
- public static int enumerate(Thread[] tarray)
- public Thread.State getState()
- public final ThreadGroup getThreadGroup()
- public String toString()
- public final void notify()
- public final void notifyAll()
- public void setContextClassLoader(ClassLoader cl)
- public ClassLoader getContextClassLoader()
- public static Thread.UncaughtExceptionHandler getDefaultUncaughtExceptionHandler()
- public static void setDefaultUncaughtExceptionHandler(Thread.UncaughtExceptionHandler eh)

Thread Creation

While multithreading in Java, you can create a thread using two ways:

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

What is Thread Class?

Thread class provides the methods and constructors to create and perform operations on a thread. Thread class extends Object class and implements the Runnable interface.

Various constructors are used in a Thread class, but the commonly used constructors are:

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

Also, as discussed earlier, there are various thread methods that are used for different purposes and tasks.

So, these constructors and methods are provided by the Thread class to perform various operations on a thread.

What is a Runnable Interface?

Runnable Interface is implemented whose instances are intended to be executed by a thread. It has only one method run().

public void run() – This is used to perform an action for a thread.

Starting a Thread

While multithreading in Java, to start a newly created thread, the start() method is used.

- A new thread starts(with a new callstack).
- The thread moves from the New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Java Thread Example by extending Thread Class

```
1
2   class Multi extends Thread{
3   public void run(){
4   System.out.println("thread is running...");
5   }
6   public static void main(String args[]){
7   Multi thread1=new Multi();
8   thread1.start();
    }
9   }
```

Output:

thread is running…

Java Thread Example by implementing Runnable interface

```
1
2     class Multi3 implements Runnable{
3     public void run(){
4     System.out.println("thread is running...");
5     }
6
7     public static void main(String args[]){
8     Multi3 m1=new Multi3();
9     Thread thread1 =new Thread(m1);
10    thread1.start();
11     }
      }
```

Output: