Java.lang.ThreadGroup class in Java

Difficulty Level : Medium Last Updated : 04 Jul, 2017

ThreadGroup creates a group of threads. It offers a convenient way to manage groups of threads as a unit. This is particularly valuable in situation in which you want to suspend and resume a number of related threads.

- The thread group form a tree in which every thread group except the initial thread group has a parent.
- A thread is allowed to access information about its own thread group but not to access information about its thread group's parent thread group or any other thread group.

Constructors:

1. **public ThreadGroup(String name):** Constructs a new thread group. The parent of this new group is the thread group of the currently running thread.

```
Throws: SecurityException - if the current thread cannot
 create a thread in the specified thread group.
```

2. **public ThreadGroup(ThreadGroup parent, String name):** Creates a new thread group. The parent of this new group is the specified thread group.

```
Throws:
NullPointerException - if the thread group argument is null.
SecurityException - if the current thread cannot create a thread in the
specified thread group.
```

Methods

1. **int activeCount():** This method returns the number of threads in the group plus any group for which this thread is parent.

```
Syntax: public int activeCount()
Returns: This method returns an estimate of the number of
active threads in this thread group and in any other thread group
that has this thread group as an ancestor.
Exception: NA
```

```java
// Java code illustrating activeCount() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 1000; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[])
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("parent thread group");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");

        // checking the number of active thread
        System.out.println("number of active thread: "
                           + gfg.activeCount());
    }
}
```

Output:

```
Starting one
Starting two
number of active thread: 2
```

2. **int activeGroupCount():** This method returns an estimate of the number of active groups in this thread group.

```
Syntax: public int activeGroupCount().
Returns: Returns the number of groups for which the invoking thread is pare
Exception: NA.
```

```
// Java code illustrating activeGroupCount() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 1000; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
            " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("gfg");

        ThreadGroup gfg_child = new ThreadGroup(gfg, "child");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");

        // checking the number of active thread
        System.out.println("number of active thread group: "
                                  + gfg.activeGroupCount());
    }
}
```

Output:

```
Starting one
Starting two
number of active thread group: 2
one finished executing
two finished executing
```

3. **void checkAccess():** Causes the security manager to verify that the invoking thread may access and/ or change the group on which **checkAccess()** is called.

```
Syntax: final void checkAccess().
Returns: NA.
Exception: NA.
```

```
// Java code illustrating checkAccess() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 1000; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
              " finished executing");
    }
```

```java
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");

        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");
        gfg.checkAccess();
        System.out.println(gfg.getName() + " has access");
        gfg_child.checkAccess();
        System.out.println(gfg_child.getName() + " has access");
    }
}
```

Output:

```
Starting one
Starting two
Parent thread has access
child thread has access
one finished executing
two finished executing
```

4. **void destroy():** Destroys the thread group and any child groups on which it is called.

```
Syntax: public void destroy().
Returns: NA.
Exception:
IllegalThreadStateException - if the thread group is not
empty or if the thread group has already been destroyed.
SecurityException - if the current thread cannot modify this thread group.
```

```java
// Java code illustrating destroy() method
import java.lang.*;
```

```
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");

        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");

        // block until other thread is finished
        t1.join();
        t2.join();

        // destroying child thread
        gfg_child.destroy();
        System.out.println(gfg_child.getName() + " destroyed");

        // destroying parent thread
        gfg.destroy();
        System.out.println(gfg.getName() + " destroyed");
    }
}
```

Output:

```
Starting one
Starting two
child thread destroyed
Parent thread destroyed
```

5. **int enumerate(Thread group[]):** The thread that comprise the invoking thread group are put into the group array.

```
Syntax: public int enumerate(Thread group[]).
Returns: the number of threads put into the array.
Exception: SecurityException - if the current thread
does not have permission to enumerate this thread group.
```

```java
// Java code illustrating enumerate() method.
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex) {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
             " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");

        // returns the number of threads put into the array
        Thread[] group = new Thread[gfg.activeCount()];
        int count = gfg.enumerate(group);
        for (int i = 0; i < count; i++)
        {
            System.out.println("Thread " + group[i].getName() + " found");
        }
    }
}
```

Output:

```
Starting one
Starting two
Thread one found
Thread two found
one finished executing
two finished executing
```

6. **int enumerate(Thread[] group, boolean recurse):** The threads that comprise the invoking thread group are put into the group array. If all is **true**, then threads in all subgroups of the thread are also put into group.

   Syntax: public int enumerate(Thread[] list, boolean recurse).
   Returns: the number of threads placed into the array.
   Exception: SecurityException - if the current thread does
   not have permission to enumerate this thread group.

```
// Java code illustrating enumerate(Thread[] group, boolean recurse)
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
```

```
            start();
        }
    public void run()
        {

            for (int i = 0; i < 10; i++)
            {
                try
                {
                    Thread.sleep(10);
                }
                catch (InterruptedException ex)
                {
                    System.out.println("Exception encounterted");
                }
            }
            System.out.println(Thread.currentThread().getName() +
                  " finished executing");
        }
    }
    public class ThreadGroupDemo
    {
        public static void main(String arg[]) throws InterruptedException,
            SecurityException
        {
            // creating the thread group
            ThreadGroup gfg = new ThreadGroup("Parent thread");

            ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

            NewThread t1 = new NewThread("one", gfg);
            System.out.println("Starting one");
            NewThread t2 = new NewThread("two", gfg);
            System.out.println("Starting two");

            // returns the number of threads put into the array
            Thread[] group = new Thread[gfg.activeCount()];
            int count = gfg.enumerate(group, true);
            for (int i = 0; i < count; i++)
            {
                System.out.println("Thread " + group[i].getName() + " found");
            }
        }
    }
```

Output:

```
Starting one
Starting two
Thread one found
Thread two found
one finished executing
two finished executing
```

7. **int enumerate(ThreadGroup[] group):** The subgroups of the evoking thread group are put into the group array.

```
Syntax: public int enumerate(ThreadGroup[] group).
Returns: the number of thread groups put into the array.
Exception: SecurityException - if the current thread does
not have permission to enumerate this thread group.
```

```java
// Java code illustrating enumerate(ThreadGroup[] group) method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
               " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");

        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");
        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting two");

        // returns the number of threads put into the array
        ThreadGroup[] group = new ThreadGroup[gfg.activeCount()];
        int count = gfg.enumerate(group);
        for (int i = 0; i < count; i++)
        {
            System.out.println("ThreadGroup " + group[i].getName() +
                " found");
        }
    }
}
```

Output:

```
Starting one
Starting two
ThreadGroup child thread found
two finished executing
one finished executing
```

8. **int enumerate(ThreadGroup[] group, boolean all):** The subgroups of the invoking thread group are put into the group array. If all is **true**, then all subgroups of the subgroups(and so on) are also put into group.

```
Syntax: public int enumerate(ThreadGroup[] group, boolean all)
Returns: the number of thread groups put into the array.
Exception: SecurityException - if the current thread does
not have permission to enumerate this thread group.
```

```
// Java code illustrating enumerate(ThreadGroup[] group, boolean all)
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
```

```
        {

            for (int i = 0; i < 10; i++)
            {
                try
                {
                    Thread.sleep(10);
                }
                catch (InterruptedException ex)
                {
                    System.out.println("Exception encounterted");
                }
            }
            System.out.println(Thread.currentThread().getName() +
                   " finished executing");
        }
    }
    public class ThreadGroupDemo
    {
        public static void main(String arg[]) throws InterruptedException,
            SecurityException
        {
            // creating the thread group
            ThreadGroup gfg = new ThreadGroup("Parent thread");

            ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

            NewThread t1 = new NewThread("one", gfg);
            System.out.println("Starting one");
            NewThread t2 = new NewThread("two", gfg);
            System.out.println("Starting two");

            // returns the number of threads put into the array
            ThreadGroup[] group = new ThreadGroup[gfg.activeCount()];
            int count = gfg.enumerate(group, true);
            for (int i = 0; i < count; i++)
            {
                System.out.println("ThreadGroup " + group[i].getName() +
                    " found");
            }
        }
    }
```

Output:

```
Starting one
Starting two
ThreadGroup child thread found
two finished executing
one finished executing
```

9. **int getMaxPriority():** Returns the maximum priority setting for the group.

```
Syntax: final int getMaxPriority().
Returns: the maximum priority that a thread in this thread group can have.
Exception: NA.
```

```java
// Java code illustrating getMaxPriority() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
               " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        // checking the maximum priority of parent thread
        System.out.println("Maximum priority of ParentThreadGroup = "
                           + gfg.getMaxPriority());

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting one");
        NewThread t2 = new NewThread("two", gfg);
```

```
            System.out.println("Starting two");
        }
    }
```

Output:

```
Maximum priority of ParentThreadGroup = 10
Starting one
Starting two
two finished executing
one finished executing
```

10. **String getName():** This method returns the name of the group.

```
Syntax: final String getName().
Returns: the name of this thread group.
Exception: NA.
```

```
// Java code illustrating getName() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
             " finished executing");
    }
}
```

```
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());
    }
}
```

Output:

```
Starting one
Starting two
two finished executing
one finished executing
```

11. **ThreadGroup getParent():** Returns null if the invoking ThreadGroup object has no parent. Otherwise, it returns the parent of the invoking object.

```
Syntax: final ThreadGroup getParent().
Returns: the parent of this thread group.
The top-level thread group is the only thread group
whose parent is null.
Exception: SecurityException - if the current thread
cannot modify this thread group.
```

```
// Java code illustrating getParent() method
import java.lang.*;
class NewThread extends Thread {
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {
        for (int i = 0; i < 10; i++) {
            try {
                Thread.sleep(10);
            }
            catch (InterruptedException ex) {
                System.out.println("Exception encounterted");
            }
        }
        System.out.println(Thread.currentThread().getName()
                                                   + " finished executing");
    }
} public class ThreadGroupDemo {
public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        // prints the parent ThreadGroup
        // of both parent and child threads
        System.out.println("ParentThreadGroup for " + gfg.getName() +
                        " is " + gfg.getParent().getName());
        System.out.println("ParentThreadGroup for " + gfg_child.getName()
                        + " is " + gfg_child.getParent().getName());
    }
}
```

Output:

```
Starting one
Starting two
ParentThreadGroup for Parent thread is main
ParentThreadGroup for child thread is Parent thread
one finished executing
two finished executing
```

12. **void interrupt():** Invokes the **interrupt()** methods of all threads in the group.

```
Syntax: public final void interrupt()
```

```
Syntax: public final void interrupt().
Returns: NA.
Exception: SecurityException - if the current thread is not
allowed to access this thread group or any of the threads in the thread gr
```

```
// Java code illustrating interrupt() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                     + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
             " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        // interrupting thread group
        gfg.interrupt();
    }
}
```

Output:

```
Starting one
Starting two
Thread two interrupted
Thread one interrupted
one finished executing
two finished executing
```

13. **boolean isDaemon():** Tests if this thread group is a daemon thread group. A daemon thread group is automatically destroyed when its last thread is stopped or its last thread group is destroyed.

    Syntax: public final boolean isDaemon().
    Returns: **true** if the group is daemon group. Otherwise it returns false.
    Exception: NA.

```
// Java code illustrating isDaemon() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                    + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
                " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        if (gfg.isDaemon() == true)
            System.out.println("Group is Daemon group");
        else
            System.out.println("Group is not Daemon group");
    }
}
```

Output:

```
Starting one
Starting two
Group is not Daemon group
two finished executing
one finished executing
```

14. **boolean isDestroyed():** This method tests if this thread group has been destroyed.

```
Syntax: public boolean isDestroyed().
Returns: true if this object is destroyed.
Exception: NA.
```

```
// Java code illustrating isDestroyed() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                     + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
                " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException, Exception
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        if (gfg.isDestroyed() == true)
            System.out.println("Group is destroyed");
        else
            System.out.println("Group is not destroyed");
    }
}
```

Output:

```
Starting one
Starting two
Group is not destroyed
one finished executing
two finished executing
```

15. **void list():** Displays information about the group.

    Syntax: public void list().

    Returns: NA.

    Exception: NA.

```java
// Java code illustrating list() method.
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                     + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
              " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException, Exception
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        // listing contents of parent ThreadGroup
        System.out.println("\nListing parentThreadGroup: " + gfg.getName()
              + ":");
        // prints information about this thread group
        // to the standard output
        gfg.list();
    }
}
```

Output:

```
Starting one
Starting two

Listing parentThreadGroup: Parent thread:
java.lang.ThreadGroup[name=Parent thread, maxpri=10]
    Thread[one, 5, Parent thread]
    Thread[two, 5, Parent thread]
    java.lang.ThreadGroup[name=child thread, maxpri=10]
one finished executing
two finished executing
```

16. **boolean parentOf(ThreadGroup group):** This method tests if this thread group is either the thread group argument or one of its ancestor thread groups.

```
Syntax: final boolean parentOf(ThreadGroup group).
Returns: true if the invoking thread is the parent
of group(or group itself). Otherwise, it returns false.
Exception: NA.
```

```
// Java code illustrating parentOf() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
```

```
                {
                    Thread.sleep(10);
                }
                catch (InterruptedException ex)
                {
                  System.out.println("Thread " + Thread.currentThread().getName
                                         + " interrupted");
                }
            }
            System.out.println(Thread.currentThread().getName() +
                    " finished executing");
        }
    }
    public class ThreadGroupDemo
    {
        public static void main(String arg[]) throws InterruptedException,
            SecurityException, Exception
        {
            // creating the thread group
            ThreadGroup gfg = new ThreadGroup("Parent thread");
            ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

            NewThread t1 = new NewThread("one", gfg);
            System.out.println("Starting " + t1.getName());
            NewThread t2 = new NewThread("two", gfg);
            System.out.println("Starting " + t2.getName());

            // checking who is parent thread
            if (gfg.parentOf(gfg_child))
                System.out.println(gfg.getName() + " is parent of " +
                   gfg_child.getName());
        }
    }
```

Output:

```
Starting one
Starting two
Parent thread is parent of child thread
two finished executing
one finished executing
```

17. **void setDaemon(boolean isDaemon):** This method changes the daemon status of this thread group. A daemon thread group is automatically destroyed when its last thread is stopped or its last thread group is destroyed.

```
Syntax: final void setDaemon(boolean isDaemon).
Returns: If isDaemon is true, then the invoking group is
```

flagged as a daemon group.
Exception: SecurityException - if the current thread cannot modify this thread group.

```
// Java code illustrating setDaemon() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                     + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
               " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException, Exception
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        // daemon status is set to true
        gfg.setDaemon(true);

        // daemon status is set to true
        gfg_child.setDaemon(true);
        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());
        if (gfg.isDaemon() && gfg_child.isDaemon())
            System.out.println("Parent Thread group and "
                               + "child thread group"
                               + " is daemon");
    }
}
```

Output:

```
Starting one
Starting two
Parent Thread group and child thread group is daemon
one finished executing
two finished executing
```

18. **void setMaxPriority(int priority):** Sets the maximum priority of the invoking group to priority.

```
Syntax: final void setMaxPriority(int priority).
Returns: NA.
Exception: SecurityException - if the current thread
cannot
modify this thread group.
```

```
// Java code illustrating setMaxPriority() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                            + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
                 " [priority = " +
           Thread.currentThread().getPriority() + "]
                finished executing.");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException, Exception
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");
        gfg.setMaxPriority(Thread.MAX_PRIORITY - 2);
        gfg_child.setMaxPriority(Thread.NORM_PRIORITY);

        NewThread t1 = new NewThread("one", gfg);
        t1.setPriority(Thread.MAX_PRIORITY);
        System.out.println("Starting " + t1.getName());
        t1.start();
        NewThread t2 = new NewThread("two", gfg_child);
        t2.setPriority(Thread.MAX_PRIORITY);
        System.out.println("Starting " + t2.getName());
        t2.start();
    }
}
```

Output:

```
Starting one
Starting two
two [priority = 5] finished executing.
one [priority = 8] finished executing.
```

19. **String toString():** This method returns a string representation of this Thread group.

```
Syntax: public String toString().
Returns: String equivalent of the group.
Exception: SecurityException - if the current thread
cannot modify this thread group.
```

```java
// Java code illustrating toString() method
import java.lang.*;
class NewThread extends Thread
{
    NewThread(String threadname, ThreadGroup tgob)
    {
        super(tgob, threadname);
        start();
    }
public void run()
    {

        for (int i = 0; i < 10; i++)
        {
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException ex)
            {
              System.out.println("Thread " + Thread.currentThread().getName
                                          + " interrupted");
            }
        }
        System.out.println(Thread.currentThread().getName() +
              " finished executing");
    }
}
public class ThreadGroupDemo
{
    public static void main(String arg[]) throws InterruptedException,
        SecurityException, Exception
    {
        // creating the thread group
        ThreadGroup gfg = new ThreadGroup("Parent thread");
        ThreadGroup gfg_child = new ThreadGroup(gfg, "child thread");

        // daemon status is set to true
        gfg.setDaemon(true);

        // daemon status is set to true
        gfg_child.setDaemon(true);
        NewThread t1 = new NewThread("one", gfg);
        System.out.println("Starting " + t1.getName());
        NewThread t2 = new NewThread("two", gfg);
        System.out.println("Starting " + t2.getName());

        // string equivalent of the parent group
        System.out.println("String equivalent: " + gfg.toString());
    }
}
```

Output:

```
Starting one
Starting two
String equivalent: java.lang.ThreadGroup[name=Parent thread, maxpri=10]
one finished executing
two finished executing
```