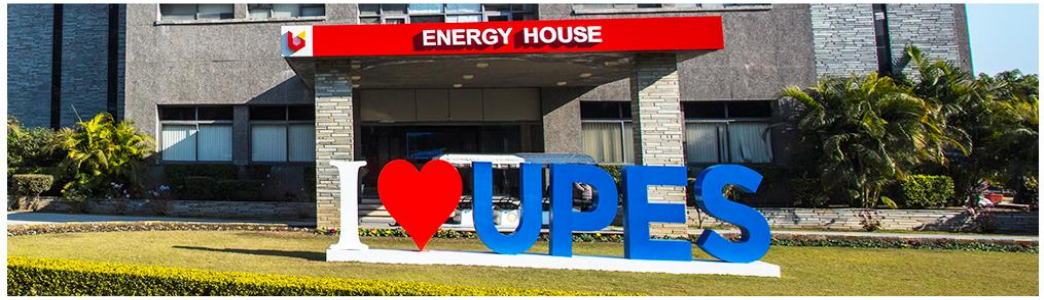



# UPES

UNIVERSITY WITH A PURPOSE

# Thread Management, Security, and ThreadGroup

- Threads are organized into thread groups for management and security reasons.
- Every Java thread is a member of a *thread group*. Thread groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once, rather than individually. For example, you can start or suspend all the threads within a group with a single method call.
- Threads within a group can be managed as a unit.
- The thread group can also be used to define a security domain.
- Java thread groups are implemented by the **ThreadGroup** class in the **java.lang** package.

# ThreadGroup

- The runtime system puts a thread into a thread group during thread construction.

**The Default Thread Group**

- If you create a new Thread without specifying its group in the constructor, the runtime system automatically places the new thread in the same group as the thread that created it.

**Creating a Thread Explicitly in a Group**

**Exp:**

```
ThreadGroup myThreadGroup = new ThreadGroup("My Group of Threads");
Thread myThread = new Thread(myThreadGroup, "a thread for my group");
```

UPES
UNIVERSITY WITH A PURPOSE

# ThreadGroup (contd..)

**Getting a Thread's Group**

- To find out what group a thread is in, you can call its getThreadGroup method:

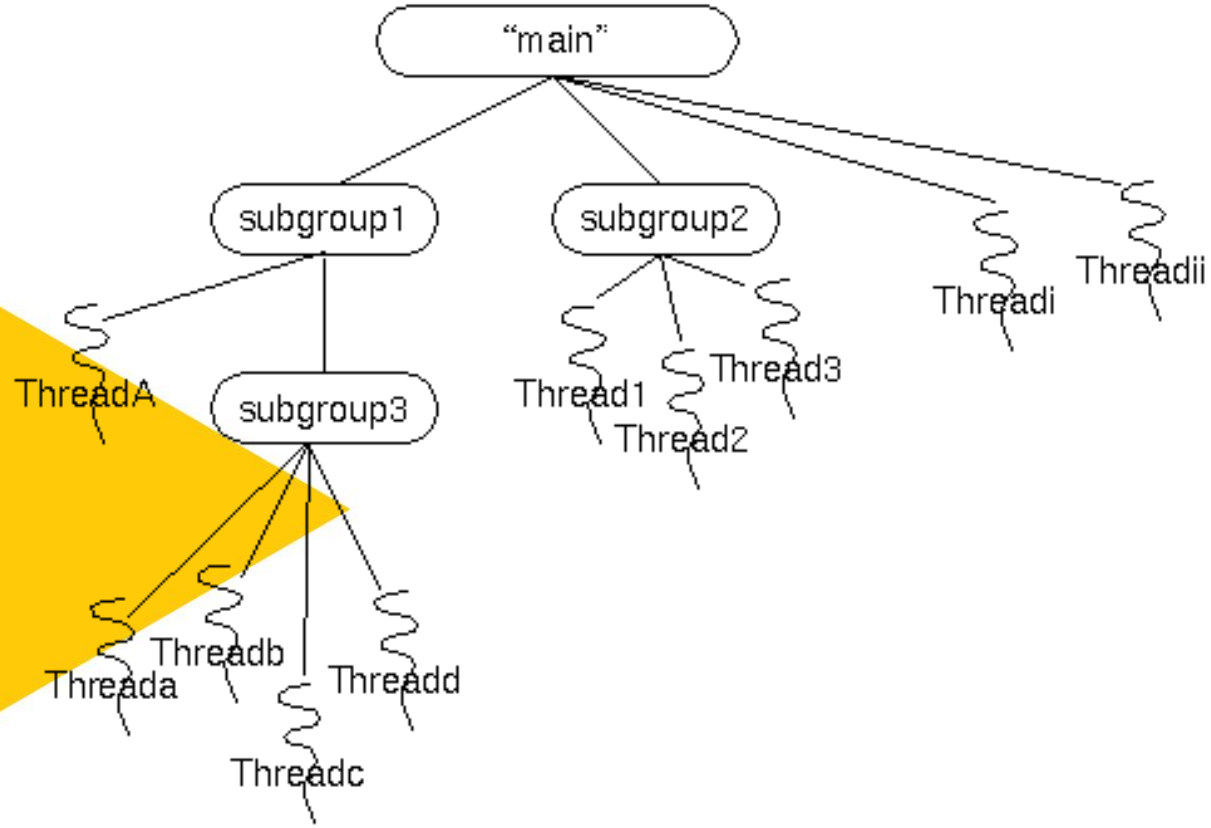
```
theGroup = myThread.getThreadGroup();
```

**The ThreadGroup Class**

- Once you've obtained a thread's ThreadGroup, you can query the group for information, such as what other threads are in the group and modify the threads.
- The ThreadGroup class manages groups of threads for Java applications.
- ThreadGroups can contain not only threads but also other ThreadGroups. The top-most thread group in a Java application is the thread group named **main**.

# ThreadGroup (contd..)

- You can create threads and thread groups in the main group.
- You can also create threads and thread groups in subgroups of main. The result is a root-like hierarchy of threads and thread groups:




UPES
UNIVERSITY WITH A PURPOSE

# Methods of ThreadGroup

**Constructor Detail:**

`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.

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

# Methods of ThreadGroup

**Methods Detail:**

1. `public final String getName()` → Returns the name of this thread group.
2. `public final ThreadGroup getParent()` → Returns the parent of this thread group.
3. `public final int getMaxPriority()` → Returns the maximum priority of this thread group. Threads that are part of this group cannot have a higher priority than the maximum priority.
4. `public final 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.
5. `public boolean isDestroyed()` → Tests if this thread group has been destroyed.
6. `public final void setMaxPriority(int pri)` → Sets the maximum priority of the group. Threads in the thread group that already have a higher priority are not affected.

# Methods of ThreadGroup

**Methods Detail:**

`7. public final boolean parentOf(ThreadGroup g)` → Tests if this thread group is either the thread group argument or one of its ancestor thread groups.

`8. public int activeCount()` → Returns an estimate of the number of active threads in this thread group and its subgroups. Recursively iterates over all subgroups in this thread group.

`9. public int activeGroupCount()` → Returns an estimate of the number of active groups in this thread group and its subgroups. Recursively iterates over all subgroups in this thread group.

`10. @Deprecated`

`public final void suspend()` → Deprecated. This method is inherently deadlock-prone. Suspends all threads in this thread group.

UPES
UNIVERSITY WITH A PURPOSE

# Methods of ThreadGroup

**Methods Detail:**

11. **`@Deprecated`**

`public final void resume()` → This method is used solely in conjunction with `Thread.suspend` and `ThreadGroup.suspend`, both of which have been deprecated, as they are inherently deadlock-prone. Resumes all threads in this thread group.

12. `public final void destroy()` → Destroys this thread group and all of its subgroups. This thread group must be empty, indicating that all threads that had been in this thread group have since stopped.

# Debugging Threads

- A few Thread methods are designed to help you debug a multithreaded application. These debugging aids can be used to print the state of an application.
- We can invoke these methods on a Thread object for debugging the threads:

1. `public String toString()` → Returns a string representation of the thread, including its name, its priority, and the name of its thread group.
2. `public long getId()` → Returns a positive value that uniquely identifies this thread while it is alive.
3. `public Thread.State getState()` → Returns the current state of this thread.
4. `public static void dumpStack()` → Prints a stack trace for the current thread on `System.err`.
5. `public String toString()` → Returns a string representation of the `ThreadGroup`, including its name and priority.
6. `public void list()` → Lists this `ThreadGroup` recursively to `System.out`. This prints the `toString` value for each of the threads and thread groups within this group.

UPES
UNIVERSITY WITH A PURPOSE

# References


- Schildt, H. (2014). *Java: the complete reference*. McGraw-Hill Education Group.
- https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
- http://journals.ecs.soton.ac.uk/java/tutorial/java/threads/group.html


UPES
UNIVERSITY WITH A PURPOSE

# THANK YOU

UPES

UNIVERSITY WITH A PURPOSE