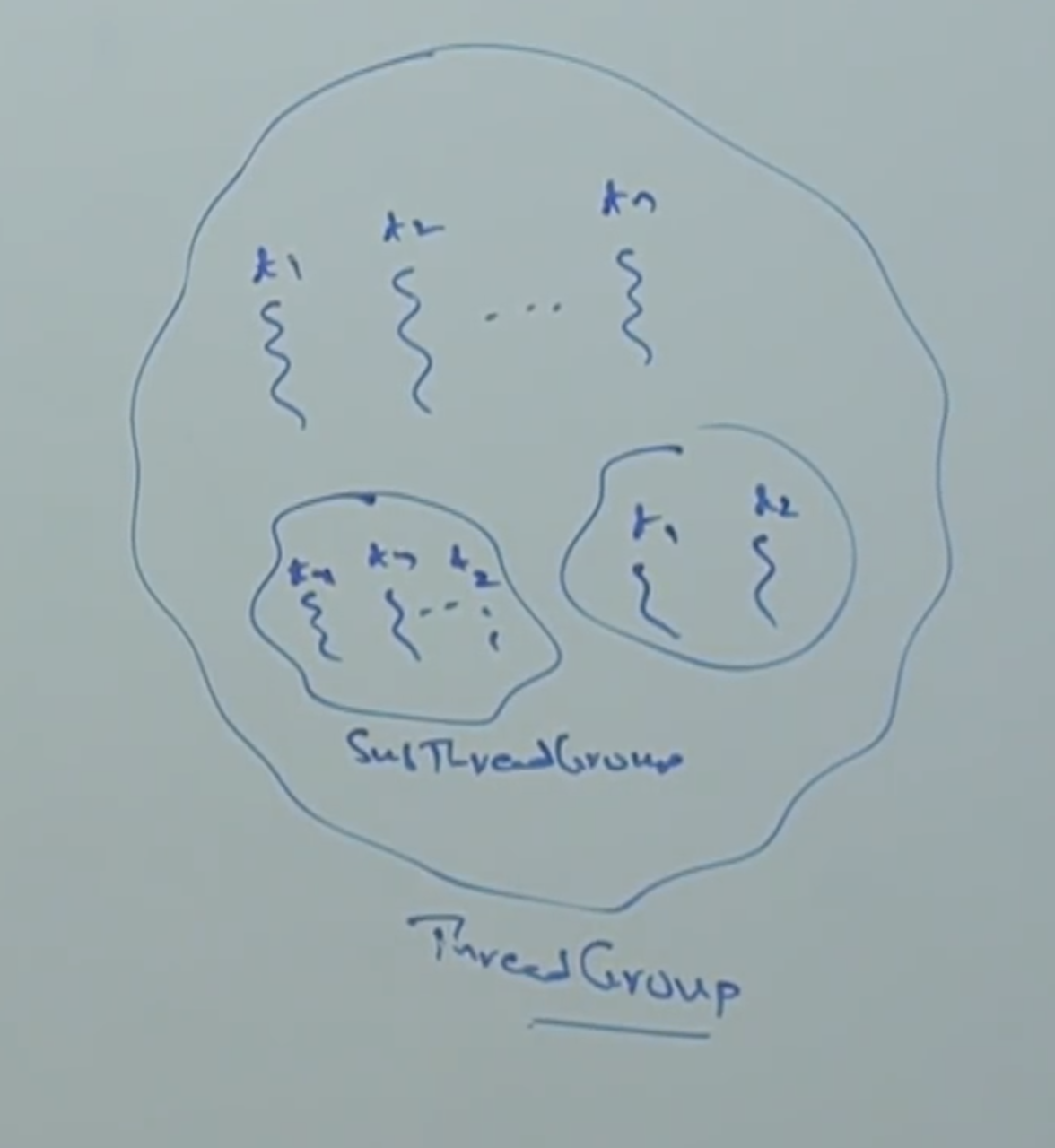
**Multithreading Enhancements – Part-01**

* **Thread Group:**

Based on functionality we can group threads into a single unit, which is nothing but thread group. That is thread group contains a group of threads.

In addition to threads thread group can also contain sub thread groups.

Example:



The main advantage of maintaining threads in the form of thread group is, we can perform common operations very easily.

Good example:

In WhatsApp we are having a group, if you want to send one message to a group of people, we can send in a single shot. Wherein in normal way we have to select all the contacts manually and send.

Every thread in Java belongs to some group, main thread belongs to “main group”. Every thread group in Java is the child group of “system group” either directly or in-directly. Hence, system group acts as a root for all thread groups in Java.

System group contains several system level threads like,

Finalizer

Reference Handler

Signal Dispatcher.

Attach Listener.

Example:

class Test{

public static void main(String[] args){

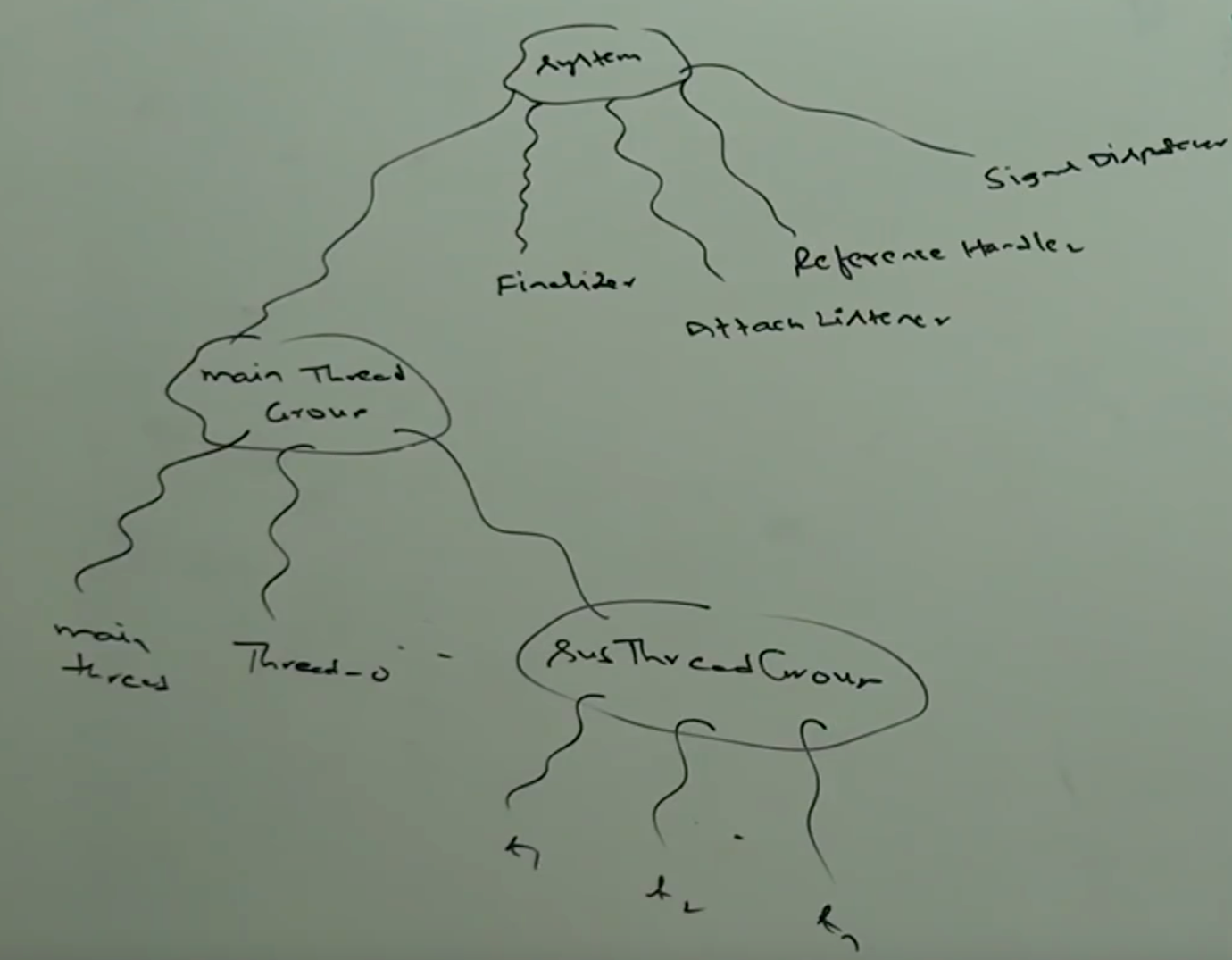
System.out.println(Thread.currentThread().getThreadGroup().getName()); // main

System.out.println(Thread.currentThread().getThreadGroup().getParent().getName()); // system.

}

}

Diagrammatic Representation:



* **Package:**

ThreadGroup is a Java class present in java.lang package and it is the direct child class of Object.

* **Constructors:**

**Constructor\_01:**

ThreadGroup g = new ThreadGroup(String name);

Creates a new thread group with a specified group name.

The parent of this new group is , the ThreadGroup of currently executing thread.

Example:

ThreadGroup g = new ThreadGroup(“First Group”);

class Test{

public static void main(String[] args){

ThreadGroup g = new ThreadGroup(“First Group”);

System.out.println(g.getParent().getName());

}

}

**Constructor\_02:**

ThreadGroup g = new ThreadGroup(ThreadGroup pg, String groupName);

Creates a new thread group with the specified group name.

The parent of this new thread group is specified parent group.

ThreadGroup g1 = new ThreadGroup(g, “Second Group”);

Example:

class Test{

public static void main(String[] args){

ThreadGroup g = new ThreadGroup(“First Group”);

System.out.println(g.getParent().getName());

ThreadGroup g1 = new ThreadGroup(g, “Second Group”);

System.out.println(g1.getParent().getName());

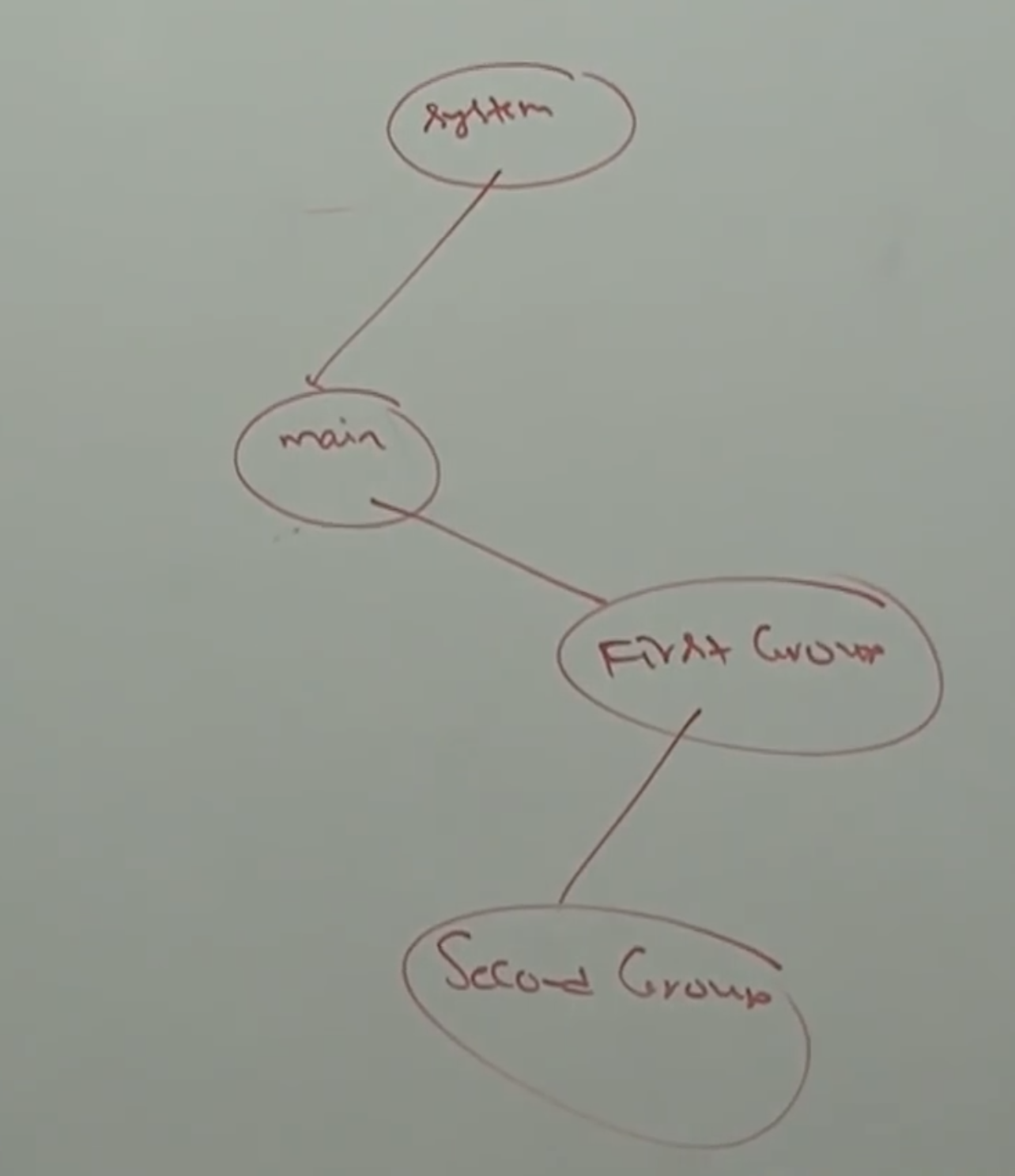
}

}

Output:

main

First Group



* **Import methods of ThreadGroup class:**

String getName()

Returns name of the ThreadGroup.

int getMaxPriority()

Returns MAX\_PRIORITY of thread group.

void setMaxPriority(int p)

To set maximum priority of thread group.

The default max priority is 10.

Note:

Threads in the thread group that have already higher priority won’t be affected but for newly added threads this max priority is applicable.

Example:

class ThreadGroupMaxPriorityDemo{

public static void main(String[] args){

ThreadGroup g = new ThreadGroup(“First Group”);

Thread t1 = new Thread(g, “Thread1”);

Thread t2 = new Thread(g, “Thread2”);

g.setMaxPriority(3);

Thread t3 = new Thread(g, “Thread3”);

System.out.println(t1.getPriority()); //5

System.out.println(t2.getPriority()); //5

System.out.println(t3.getPriority()); //3

}

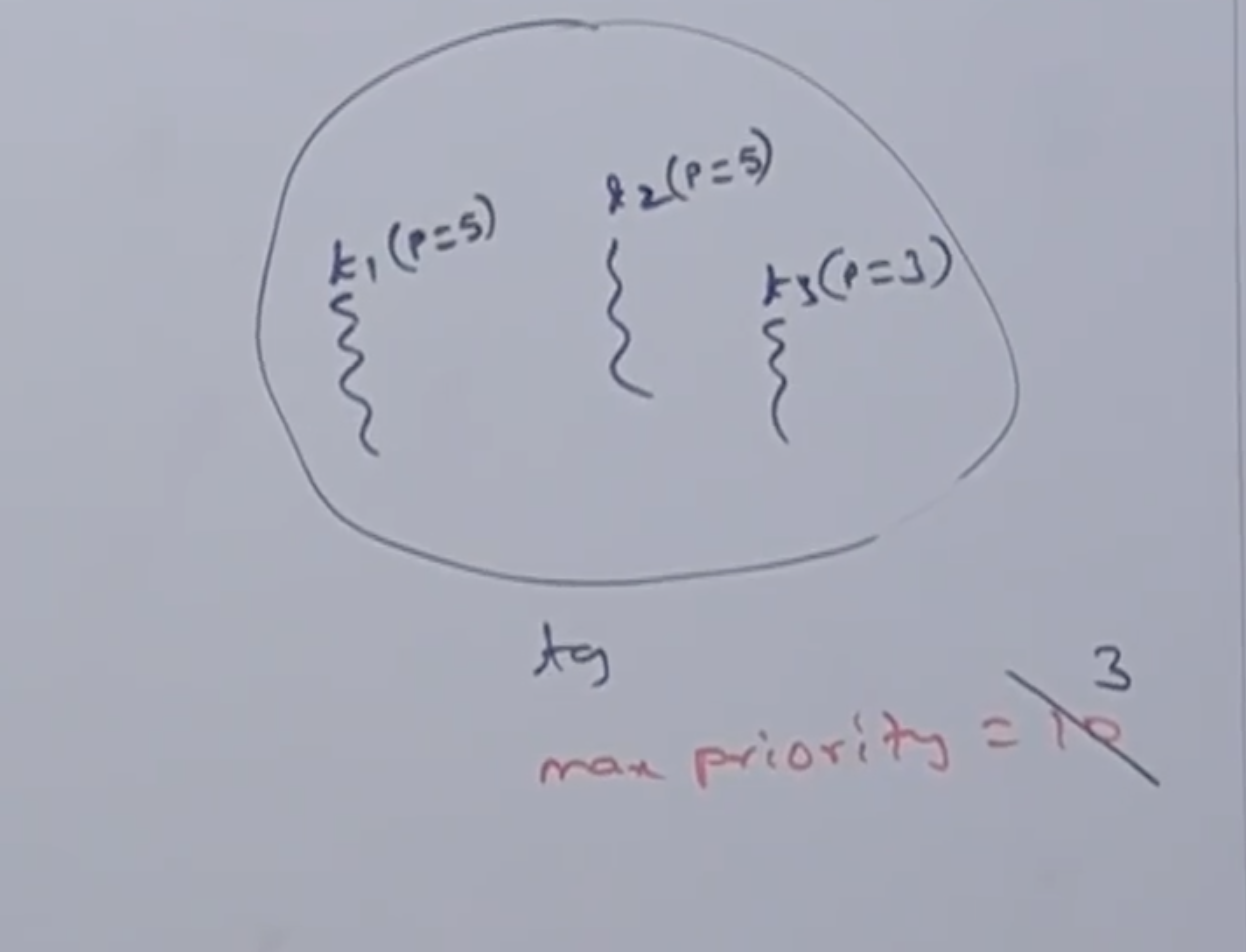
}

Output:

5

5

3



ThreadGroup getParent()

Returns parent group of current thread.

void list()

It prints information about thread group to the console.

int activeCount()

Returns number of active threads present in the thread group.

int activeGroupCount()

It returns number of active groups present in the current thread group.

int enumerate(Thread[] t)

To copy all active threads of this thread group into provided thread array.

In this case sub thread group threads also will be considered.

int enumerate(ThreadGroup[] g)

To copy all active sub thread group into thread group array.

boolean isDaemon()

To check whether the thread group is daemon or not.

void setDeamon(Boolean b)

void interrupt()

To interrupt all waiting or sleeping threads present in the thread group.

void destroy()

To destroy thread groups and its sub thread groups.

Example:

class MyThread extends Thread{

MyThread(ThreadGroup g, String name){

Super(g, name);

}

public void run(){

System.out.println(“Child Thread”);

try{

Thread.sleep

} catch(InterruptedException e){

}

}

}

class ThreadGrouDemo{

public static void main(String[] args){

ThreadGroup pg = new ThreadGroup(“ParentGroup”);

ThreadGroup cg = new ThreadGroup(pg, “ChildGroup”);

MyThread t1 = new MyThread(pg, “ChildThread1”);

MyThread t2 = new MyThread(pg, “ChildThread2”);

t1.start();

t2.start();

System.out.println(pg.activeCounts());//2

System.out.println(pg.activeGroupCount());// 1

pg.list();

Thread.sleep(10000);

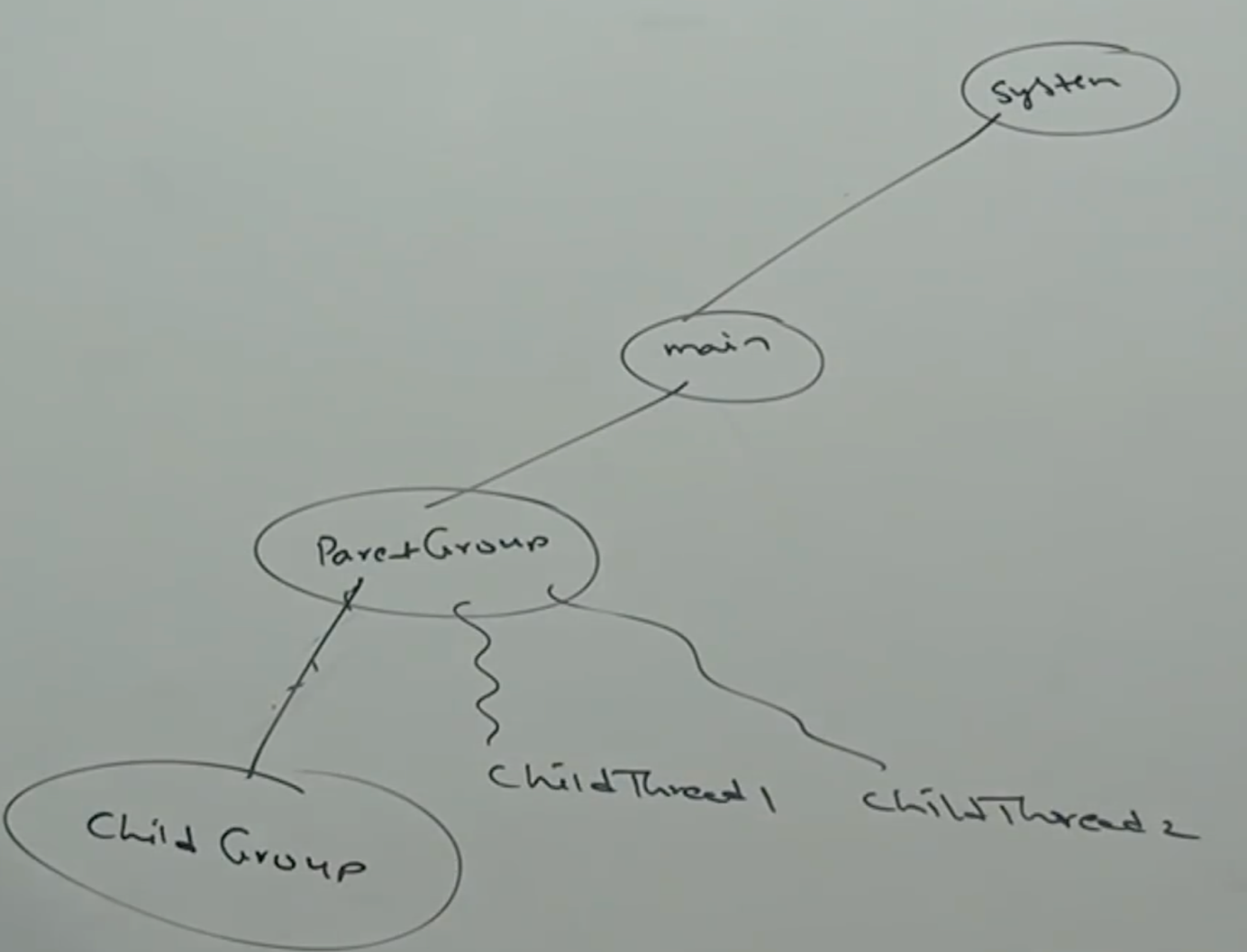
System.out.println(pg.activeCounts());//2

System.out.println(pg.activeGroupCount());// 1

pg.list();

}

}



* **Write a program to display all active thread names belongs to System and its child groups.**

class ThreadGroupDemo{

public static void main(String[] args){

ThreadGroup system =

Thread.currentThread().getThreadGroup().getParent();

Thread[] t = new Thread[system.activeCount()];

System.enumerate(t);

for(Thread t1: t){

System.out.println(t1.getName()+”—“+t1.isDaemon());

}

}

}

Output:

Reference Handler …. True

Finalizer … true

Signal Dispatcher … true

Attach Listener … true

main … false