Start will create a new thread and it will automatically call run

Sleep(2000) –static method which makes the thread sleep for 2 sec

We can create thread by runnable functional interface(in case of multiple inheritance)

To print in parallel:

**package** thread;

**class** hi **extends** Thread

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hi");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**class** hello **extends** Thread

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hello");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}}

}

}

**public** **class** threadDemo {

**public** **static** **void** main(String args[])

{

hi obj1=**new** hi();

hello obj2=**new** hello();

obj1.start();

obj2.start();

}

}

In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. Multithreading in java is a process of executing multiple activities can proceed concurrently in the same program. Thread is basically a lightweight sub-process, a smallest unit of processing. In multithreading threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources. Using two different task at the same time means multi-tasking. Thread is unit of a process.

Implementation using runnable interface:

Create a object of thread and create a runnable interface

**package** thread;

**class** hi1 **implements** Runnable

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hi");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**class** hello1 **implements** Runnable

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hello");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}}

}

}

**public** **class** threaddemorunnable {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

hi1 obj1=**new** hi1();

hello1 obj2=**new** hello1();

//since obj1 .start is not available in runnable interface

Thread t1=**new** Thread(obj1);//Runnable target

Thread t2=**new** Thread(obj2);

t1.start();

t2.start();

}

}

Join waits for t1 and t2 to complete and wait for the main thread

In the below code if t1.join and t2.join is not given bye will be executed first

**package** thread;

**class** hi1 **implements** Runnable

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hi");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**class** hello1 **implements** Runnable

{

**public** **void** run()

{

**for**(**int** i=0;i<2;i++)

{

System.***out***.println("Hello");

**try** {

Thread.*sleep*(2000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}}

}

}

**public** **class** threaddemorunnable {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

hi1 obj1=**new** hi1();

hello1 obj2=**new** hello1();

//since obj1 .start is not available in runnable interface

Thread t1=**new** Thread(obj1);//Runnable target

Thread t2=**new** Thread(obj2);

t1.start();

t2.start();

**try** {

t1.join();

t2.join();

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

System.***out***.println("bye");

}

}

Isalive is used to check whether the thread is alive

t1.setName("Hi 1"); -to set the name of the thread,by default anme is thread0

System.***out***.println("getname thread2:"+t1.getName());-to get the name of the thread

System.out.println("priority"+t1.getPriority()); to get the priority

t1.setPriority(2); -to set the priority

t1.setPriority(Thread.MIN\_PRIORITY); -sets the priority to 1

to set the priority in current thread:

System.out.println("Hi"+Thread.currentThread().getPriority());

Synchronized: //only one thread can work in the method:

**package** thread;

**class** counter

{

**int** count;

**public** **synchronized** **void** increment() //only one thread can work in the moethod

{

count++;

}

}

**public** **class** Syncclass {

**public** **static** **void** main(String args[]) **throws** Exception

{

counter c=**new** counter();

Thread t1=**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<1000;i++)

c.increment();

}

});

Thread t2=**new** Thread(**new** Runnable() {

@Override

**public** **void** run() {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<100;i++)

c.increment();

}

});

t1.start();

System.***out***.println("counter"+c.count); //prints only 0

t1.join();

System.***out***.println("counter"+c.count); //prints 100

t2.start();

System.***out***.println("counter"+c.count); //prints only 0

t2.join();

System.***out***.println("counter"+c.count); //prints 100

}

}

In Factory pattern, we create object without exposing the creation logic to the client and refer to newly created object using a common interface.