DAY-62[Multi-threading-5]

--------------------------

DAEMON THREAD:-

----------------

\* Daemon thread is considered as helping thread because it will helps the threds which is running in the application.

\* Daemon thread always executes in the background.

\* This thread will run in background from the begging to ending of the program.

example for daemon thread is "garbage collector","auto-spell check","auto-compiler","signal dispatcher"....etc:

inbuilt methods w.r.t Daemon thread.

-------------------------------------

1. public boolean isDaemon()

--> It will return the boolean result as true or false which indicates wheter the thread is daemon or non-daemon thread.

2. public void setDaemon(boolean b)

--> by using this we can make the non-daemon thread as daemon and vice-versa.

EXAMPLE:

---------

// example for daemon thread

class MyThread extends Thread

{

}

class Demo1

{

public static void main(String[] args)

{

//System.out.println(Thread.currentThread().isDaemon()); // false

//Thread.currentThread().setDaemon(true);

MyThread t1 = new MyThread();

t1.start();

System.out.println(Thread.currentThread().isDaemon()); // false

t1.setDaemon(true);

System.out.println(t1.isDaemon()); // true

}

}

EXAMPLE:

---------

// example-2 daemon thread

class MyThread extends Thread

{

public void run()

{

for (int i=0;i<=5 ;i++ )

{

System.out.println("Child thread [Daemon thread]");

try

{

Thread.sleep(2000);

}

catch (InterruptedException e)

{

}

}

}

}

class Demo2

{

public static void main(String[] args)

{

MyThread t= new MyThread();

t.setDaemon(true);

t.start();

System.out.println("Main Thread");

}

}

OUTPUT:

---------

Main Thread

Child thread [Daemon thread]

NOTE: Daemon thread ends when the all the non-daemon thread completes its execution.

but, even though the daemon threads are executing in the background jvm will destroy them immediately after the completion of non-daemon thread.

INTER-THREAD COMMUNICATIONS:

--------------------------------

when ever multi-threads are executing simultaneously in the multithreaded application there may be chance of thread dependency.

when ever thread dependency exist then there must a communication b/w these threads.

The communication b/w the threads of the same application is called as "Inter-Thread Communication".

To achive Inter thread communication we will make use of 3 methods:

1. wait()

2. notify()

3. notifyall()

\*\*\*Note: All the 3 methods are present in "object class" and not in Thread class.

wait() method:-

----------------

wait method is alternative for join method. When wait() is called the current thread goes to waiting state untill the other thread notify it.

The Thread which made another thread to wait will execute first then after executing it will notify the thread which is in wait state.

wait() is overloaded method in thread class

--> wait()

--> wait(long ms)

--> wait(long ms, int ns)

notify() method:-

-----------------

notify method is called by the executing thread to the waiting thread to indicate that thread to come out of waiting state.

\*\*\*NOTE: wait(), notify(), notifyall() all these method should be written inside synchronized block.

difference b/w notify() and notifyall() ?

--> notify() method will notify only one thread at a time. notifyall() method will notify all the thread at a time.

EXAMPLE:

--------

// example-1 interthread communication

class customer extends Thread

{

int balance =15000;

public void run()

{

for (int i=1;i<=10 ;i++ )

{

System.out.println("Transcation started..!");

try

{

Thread.sleep(2000);

}

catch (InterruptedException e)

{

}

}

for (int i=1;i<=10 ;i++ )

{

balance = balance-1000;

System.out.println("withdraw is completed..!");

try

{

Thread.sleep(2000);

}

catch (InterruptedException e)

{

}

}

synchronized (this)

{

notify();

}

for (int i=1;i<=10 ;i++ )

{

System.out.println("Transcation closed.!");

try

{

Thread.sleep(2000);

}

catch (InterruptedException e)

{

}

}

}

}

class Demo3

{

public static void main(String[] args)

{

customer c = new customer();

System.out.println("balance before the transcation is: "+c.balance);

c.start();

try

{

synchronized (c)

{

c.wait();

}

}

catch (InterruptedException e)

{

}

System.out.println("balance after the transcation is: "+c.balance);

}

}

OUTPUT:

--------

balance before the transcation is: 15000

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

Transcation started..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

withdraw is completed..!

balance after the transcation is: 5000

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

Transcation closed.!

PRODUCER AND CONSUMER PROBLEM TO EXPLAIN INTER THREAD COMMUNICATION:

---------------------------------------------------------------------

SCENARIO:

---------

In the producer and consumer problem the time taken for the production and for consumption would vary.

As long as the producer produce the item consumer can not cosume it.

In order to bring the communication b/w producer and consumer we need to use wait() and notify() method.

REFER DIA:

EXAMPLE:

--------

class Factory

{

int item = 0;

synchronized public void production() throws InterruptedException

{

++item;

System.out.println("item "+ item + "under production");

Thread.sleep(1000);

System.out.println("item"+ item + " produced");

Thread.sleep(1000);

notify();

wait();

}

synchronized public void consumption() throws InterruptedException

{

System.out.println("item"+ item + "under consumption");

Thread.sleep(1000);

System.out.println("item"+ item + " consumed");

Thread.sleep(1000);

notify();

wait();

}

}

class Producer extends Thread

{

Factory f;

public Producer(Factory f)

{

this.f=f;

}

public void run()

{

try

{

for (int i =0;i<=10 ;i++ )

{

f.production();

}

}

catch (InterruptedException e)

{

}

}

}

class Consumer extends Thread

{

Factory f;

public Consumer(Factory f)

{

this.f=f;

}

public void run()

{

try

{

for (int i =0;i<=10 ;i++ )

{

f.consumption();

}

}

catch (InterruptedException e)

{

}

}

}

class Demo4

{

public static void main(String[] args)

{

Factory f = new Factory();

Producer p = new Producer(f);

Consumer c = new Consumer(f);

p.start();

c.start();

}

}

OUTPUT:

-------

item0under consumption

item0 consumed

item 1under production

item1 produced

item1under consumption

item1 consumed

item 2under production

item2 produced

item2under consumption

item2 consumed

item 3under production

item3 produced

item3under consumption

item3 consumed

item 4under production

item4 produced

item4under consumption

item4 consumed

item 5under production

item5 produced

item5under consumption

item5 consumed

item 6under production

item6 produced

item6under consumption

item6 consumed

item 7under production

item7 produced

item7under consumption

item7 consumed

item 8under production

item8 produced

item8under consumption

item8 consumed

item 9under production

item9 produced

item9under consumption

item9 consumed

item 10under production

item10 produced

item10under consumption

item10 consumed

item 11under production

item11 produced

DEAD LOCK and STARVATION:-

---------------------------

If thread goes for waiting state for infinite time then it is called as DEAD LOCK.

If thread goes for waiting state for long period but after long time if its come to normal state then is called as STARVATION.

NOTE: generally thread will go to starvation when ever all the other threads priority is higher than its prioirty.

EXAMPLE:

--------

// example for dead lock

class A

{

public synchronized void method1(B b)

{

System.out.println("Thread 1 starts execution of class A's method 1");

try

{

Thread.sleep(100);

}

catch (InterruptedException e)

{

}

System.out.println("Thread 1 trying to call B's method2");

b.method2();

}

public synchronized void method2()

{

System.out.println("inside A this is method 2");

}

}

class B

{

public synchronized void method1(A a)

{

System.out.println("Thread 2 starts execution of class B's method 1");

try

{

Thread.sleep(5000);

}

catch (InterruptedException e)

{

}

System.out.println("Thread 2 trying to call A's method2");

a.method2();

}

public synchronized void method2()

{

System.out.println("inside B this is method2");

}

}

class Demo5 extends Thread

{

A a= new A();

B b = new B();

public void m1()

{

this.start();

a.method1(b);

}

public void run()

{

b.method1(a);

}

public static void main(String[] args)

{

Demo5 d= new Demo5();

d.m1();

}

}

OUTPUT:

----------

Thread 1 starts execution of class A's method 1

Thread 2 starts execution of class B's method 1

Thread 1 trying to call B's method2

Thread 2 trying to call A's method2