GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT302

OBJECT ORIENTED PROGRAMMING

UNIT 12

MULTITHREADED PROGRAMMING

Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- Introduction, Creating Threads, Extending Thread Class, Runnable Interface
- Stopping and blocking a thread, Life cycle of thread, Thread Methods
- Thread Exception, Thread Priority, Synchronizations

Multithreading in Java

- **Multithreading in** Java is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

Advantages of Java Multithreading

- It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.
- You **can perform many operations together, so it saves time.**
- Threads are **independent,** so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

- Multitasking is a process of executing multiple tasks simultaneously.
- We use multitasking to utilize the CPU.

Multitasking can be achieved in two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

Process-based Multitasking (Multiprocessing)

- Each process has an address in memory. In other words, each process allocates a separate memory area.
- A process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

Thread-based Multitasking (Multithreading)

- Threads share the same address space.
- A thread is lightweight.
- Cost of communication between the thread is low.

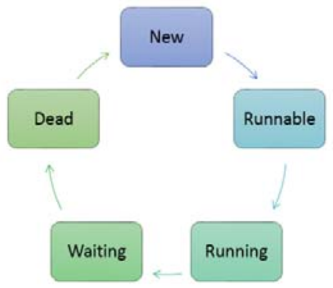
What is Thread in java

- A thread is a lightweight sub process, the smallest unit of processing. It is a separate path of execution.
- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

Life cycle of a Thread

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

- New
- Runnable
- Running
- Non-Runnable (Blocked)
- Terminated



New

The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

Runnable

The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.

Running

The thread is in running state if the thread scheduler has selected it.

Non-Runnable (Blocked)

This is the state when the thread is still alive, but is currently not eligible to run.

Terminated

A thread is in terminated or dead state when its run() method exits.

Creating Threads

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

Thread class

- Thread class provide constructors and methods to create and perform operations on a thread.
- Thread class extends Object class and implements Runnable interface.

Commonly used Constructors of Thread class:

- Thread()
- Thread(String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used Methods for threads

Method	Description
start()	This method starts the execution of the thread and JVM calls the run() method on the thread.
Sleep(int milliseconds)	This method makes the thread sleep hence the thread's execution will pause for milliseconds provided and after that, again the thread starts executing. This help in synchronization of the threads.
getName()	It returns the name of the thread.
setPriority(int newpriority)	It changes the priority of the thread.
yield ()	It causes current thread on halt and other threads to execute.

Runnable interface

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

public void run(): is used to perform action for a thread.

Starting a thread:

- **start() method** of Thread class is used to start a newly created thread. It performs following tasks: A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Java Thread Example by extending Thread class

```
class Multi extends Thread
{
      public void run()
      {
                System.out.println("thread is running...");
      }
public static void main(String args[])
{
Multi t1=new Multi();
t1.start();
 }
}
```

Output: thread is running...

Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable
{
      public void run()
      {
                System.out.println("thread is running...");
      }

public static void main(String args[])
{
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

Output: thread is running...

Sleep method

- The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

The Thread class provides two methods for sleeping a thread:

- public static void sleep(long miliseconds)throws InterruptedException
- public static void sleep(long miliseconds, int nanos)throws InterruptedException

Example of sleep method

```
class TestSleepMethod1 extends Thread
{
 public void run()
{
  for(int i=1;i<5;i++)
         {
             try
           {
               Thread.sleep(500);
           }
        Catch (InterruptedException e)
         {
         System.out.println(e);
         }
    System.out.println(i);
  }
 }
```

```
public static void main(String args[])
{
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();

  t1.start();
  t2.start();
 }
}
```

```
output
1
1
 2
 2
 3
 3
 4
 4
```

- Can we start a thread twice ...????

No. After starting a thread, it can never be started again. If you does so, an *IllegalThreadStateException* is thrown. In such case, thread will run once but for second time, it will throw exception.

```
public class Test extends Thread{
 public void run(){
   System.out.println("running...");
 }
 public static void main(String args[]){
  Test t1=new Test ();
  t1.start();
  t1.start();
 }
}
```

```
OUTPUT :
running
Exception in thread "main" java.lang.IllegalThreadStateException
```

getName(),setName(String) and getId() method

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name i.e. thread-0, thread-1 and so on. By we can change the name of the thread by using setName() method.

```
class Test extends Thread{
  public void run(){
   System.out.println("running...");
  }
 public static void main(String args[]){
  Test t1=new Test ();
  Test t2=new Test ();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());
  System.out.println("id of t1:"+t1.getId());
  t1.start();
  t2.start();
  t1.setName("UVPCE");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

Output:

```
Name of t1:Thread-0
Name of t2:Thread-1
 id of t1:8 running...
After changling name of t1:UVPCE
running...
```

currentThread() method

- The currentThread() method returns a reference to the currently executing thread object.

```
class Test extends Thread
{
 public void run(){
  System.out.println(Thread.currentThread().getName());
 }
 }
```

```
public static void main(String args[])
{
  Test t1=new Test ();
  Test t2=new Test ();

  t1.start();
  t2.start();
 }
}
```

```
Output:
Thread-0
Thread-1
```

Thread Priority

- Each thread have a priority. Priorities are represented by a number between 1 and 10.
- In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling).
- But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

3 constants defined in Thread class:

- public static int MIN_PRIORITY
- public static int NORM_PRIORITY
- public static int MAX_PRIORITY

Default priority of a thread is 5 (NORM_PRIORITY). The value of MIN_PRIORITY is 1 and the value of MAX_PRIORITY is 10.

```
class TestMultiPriority1 extends Thread
{
 public void run()
{
   System.out.println("running thread name is:"+Thread.currentThread().getName());
   System.out.println("running thread priority is:"+Thread.currentThread().getPriority());
  }
 public static void main(String args[]){
  TestMultiPriority1 m1=new TestMultiPriority1();
  TestMultiPriority1 m2=new TestMultiPriority1();
  m1.setPriority(Thread.MIN_PRIORITY);
  m2.setPriority(Thread.MAX_PRIORITY);
  m1.start();
  m2.start();
 }
}
```

Output:

```
running thread name is:Thread-0
running thread priority is:10
running thread name is:Thread-1
running thread priority is:1
```

THANK YOU!
ANY QUESTIONS?