OOP-Unit-9-Notes

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

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Java Types of Threads

1. User thread:

User threads are high priority threads that are created by the users or application. JVM will not exit until any user thread finishes its task before terminating it.

2. Daemon thread:

Daemon threads are low priority threads that are created to provide services to the user thread. These threads are only needed while user threads are running and are meant to serve them. Unlike user threads it does’nt prevent JVM from exiting once all user threads have finished their execution. These threads are used to perform background tasks such as garbage collection ,removing unwanted entries etc.

Java Thread class

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

Java Thread Methods

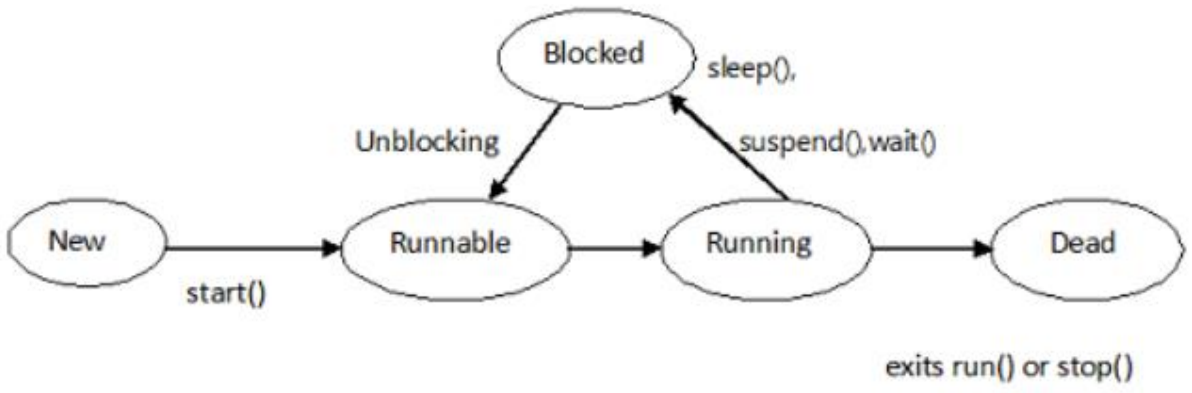
S.N.	Modifier and Type	Method	Description
1)	Void	start( )	It is used to start the execution of the thread.
2)	Void	run( )	It is used to do an action for a thread.
3)	static void	sleep( )	It sleeps a thread for the specified amount of time.
5)	Void	join( )	It is used to wait the thread to complete its execution (used in multiple thread (static method, written try catch
6)	Int	getPriority( )	It returns the priority of the thread.
7)	Void	setPriority( )	It changes the priority of the thread.
8)	String	getName( )	It returns the name of the thread.
9)	Void	setName( )	It changes the name of the thread.
10)	Long	getId( )	It returns the id of the thread.
11)	Boolean	isAlive( )	It tests if the thread is alive. True- if thread is still running False: if thread complete its execution
13)	Void	suspend()	It is used to suspend the thread.
14)	Void	resume( )	It is used to resume the suspended thread.
15)	Void	stop( )	It is used to stop the thread.

16)	Void	destroy( )	It is used to destroy the thread group and all of its subgroups.

Thread life cycle:

1. **New.**
2. **Runnable.**
3. **Running.**
4. **Blocked(Non-Runnable).**
5. **Dead.**

Diagram:



1. New: A new thread is created but not working. A thread after creation and before invocation of start() method will be in new state.

2. Runnable: A thread after invocation of start() method will be in runnable state. A thread in runnable state will be available for thread scheduler.

3. Running: A thread in execution after thread scheduler select it, it will be in running state.

4. Blocked: A thread which is alive but not in runnable or running state will be in blocked state. A thread can be in blocked state because of suspend(), sleep(), wait() methods or implicitly by JVM to perform I/O operations.

5. Dead: A thread after exiting from run() method will be in dead state. We can use stop() method to forcefully killed a thread.

Java Thread start() method

The **start()** method of thread class is used to begin the execution of thread. The result of this method is two threads that are running concurrently: the current thread (which returns from the call to the start method) and the other thread (which executes its run method).

The start() method internally calls the run() method of Runnable interface to execute the code specified in the run() method in a separate thread.

The start thread performs the following tasks:

- It stats a new thread

- The thread moves from New State to Runnable state.
- When the thread gets a chance to execute, its target run() method will run.

Java Thread run() method

The **run()** method of thread class is called if the thread was constructed using a separate Runnable object otherwise this method does nothing and returns. When the run() method calls, the code specified in the run() method is executed. You can call the run() method multiple times.

The run() method can be called using the start() method or by calling the run() method itself. But when you use run() method for calling itself, it creates problems.

Java Threads | How to create a thread in Java

There are two ways to create a thread:

1. By extending Thread class
2. 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.

//Creating Thread using inheritance (extending Thread Class)

```
class SingleThread extends Thread

{

	public void run()

	{

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

	}

}

class ST

{

	public static void main(String args[])

	{

		SingleThread st=new SingleThread ();

		st.start(); //this will implicitly call run method

	}

}
```

//Thread Creation using interface

```
import java.util.*;

class Single implements Runnable

{

        public void run()

        {

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

        }

}

class ST

{

        public static void main(String args[])

        {

                Single s=new Single();

                Thread t=new Thread(s);

//Here object of Single will be passed as a parameter in Thread class object

                t.start(); //this will implicitly call run method

        }

}
```

call the run() method more than one time

```
public class RunExp3 extends Thread
{
   public void run()
   {
      for(int i=1;i<6;i++)
      {
         try
         {
            Thread.sleep(500);
         }catch(InterruptedException e){System.out.println(e);}
      System.out.println(i);
      }
   }
   public static void main(String args[])
   {
      RunExp3 t1=new RunExp3();
      RunExp3 t2=new RunExp3();
      t1.run();
      t2.run();
   }
}
```

Java Thread suspend() method

The **suspend()** method of thread class puts the thread from running to waiting state. This method is used if you want to stop the thread execution and start it again when a certain event occurs. This method allows a thread to temporarily cease execution. The suspended thread can be resumed using the resume() method.

```
public class JavaSuspendExp extends Thread
{
   public void run()
   {
       for(int i=1; i<5; i++)
```

```
        {
            try
            {
                // thread to sleep for 500 milliseconds
                 sleep(500);
                 System.out.println(Thread.currentThread().getName());
            }catch(InterruptedException e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        // creating three threads
        JavaSuspendExp t1=new JavaSuspendExp ();
        JavaSuspendExp t2=new JavaSuspendExp ();
        JavaSuspendExp t3=new JavaSuspendExp ();
        // call run() method
        t1.start();
        t2.start();
        // suspend t2 thread
        t2.suspend();
        // call run() method
        t3.start();
    }
}
```

Java Thread stop() method

The **stop()** method of thread class terminates the thread execution. Once a thread is stopped, it cannot be restarted by start() method.

```
public class JavaStopExp extends Thread
{
    public void run()
    {
        for(int i=1; i<5; i++)
        {
            try
            {
                // thread to sleep for 500 milliseconds
                sleep(500);
                System.out.println(Thread.currentThread().getName());
            }catch(InterruptedException e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        // creating three threads
        JavaStopExp t1=new JavaStopExp ();
        JavaStopExp t2=new JavaStopExp ();
        JavaStopExp t3=new JavaStopExp ();
        // call run() method
        t1.start();
        t2.start();
        // stop t3 thread
        t3.stop();
        System.out.println("Thread t3 is stopped");     }    }
```

Java Thread join() method

The **join()** method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another. This process is like a relay race where the second runner waits until the first runner comes and hand over the flag to him.

```
public class JoinExample1 extends Thread
{
    public void run()
    {
        for(int i=1; i<=4; i++)
        {
            try
            {
                Thread.sleep(500);
            }catch(Exception e){System.out.println(e);}
            System.out.println(i);
        }
    }
    public static void main(String args[])
    {
        // creating three threads
        JoinExample1 t1 = new JoinExample1();
        JoinExample1 t2 = new JoinExample1();
        JoinExample1 t3 = new JoinExample1();
        // thread t1 starts
        t1.start();
        // starts second thread when first thread t1 is died.
        try
        {
            t1.join();
        }catch(Exception e){System.out.println(e);}
        // start t2 and t3 thread
        t2.start();
        t3.start();    }    }
```