JAVA

MULTITHREADED PROGRAMMING

By
VINOTH R
Asst. professor
Dept. of Computer Application
St.Joseph's college(Arts & Science)
Kovur, Chennai

TOPIC INCLUDES:

- Introduction to Thread
- Creation of Thread
- Life cycle of Thread
- Stopping and Blocking a Thread
- Using Thread Methods
- Thread Priority
- Thread Synchronization
- DeadLock

INTRODUCTION TO

THREAD

- **Process** and **Thread** are two basic units of Java program execution.
- **Process:** A process is a self contained execution environment and it can be seen as a program or application.
- **Thread:** It can be called *lightweight process* •
  Thread requires less resources to create and exists in the process
  - Thread shares the process resources

INTRODUCTION Contd.

Thread

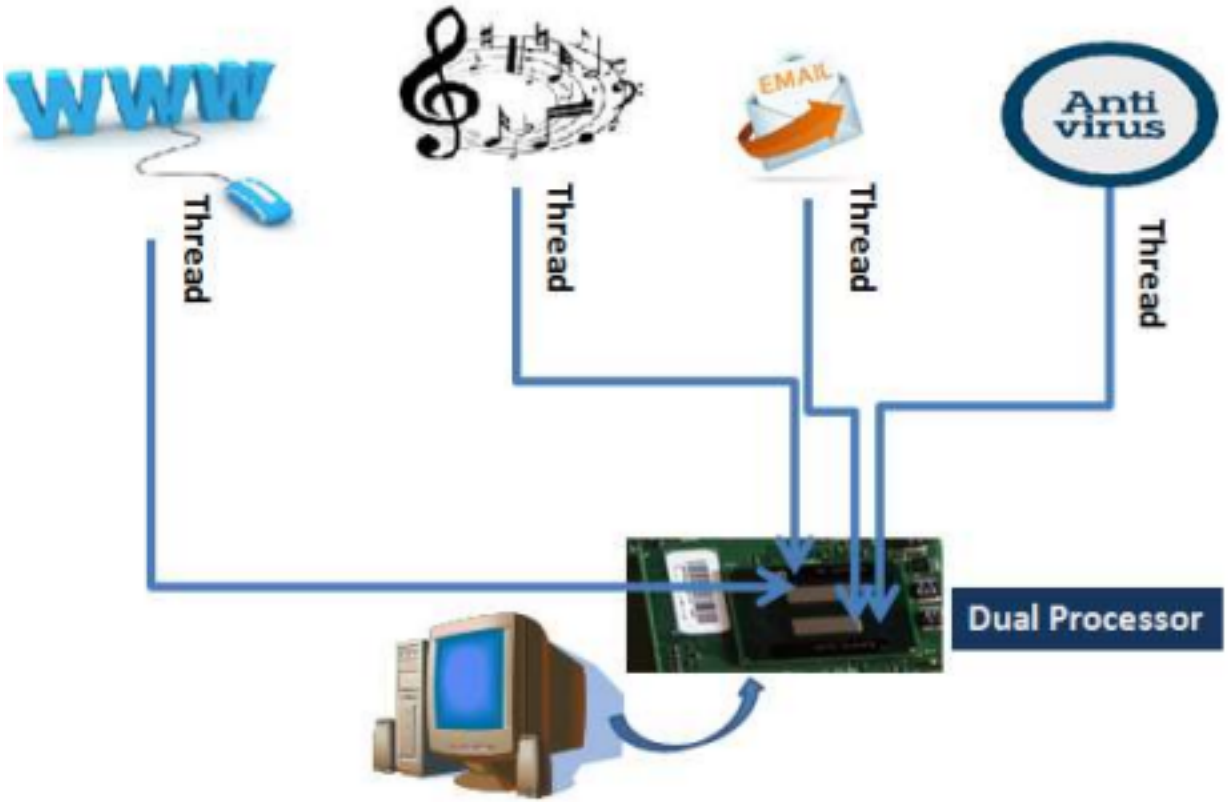
A thread is like the electricity passing through the wire to the devices.
A single wire or Multi wire can power the devices to run.

MULTITHREADING

- **Multithreading in java** is a process of executing multiple processes simultaneously
- A program is divided into two or more subprograms, which can be implemented at the same time in parallel.
- Multiprocessing and multithreading, both are used to achieve multitasking.
- Java Multithreading is mostly used in games, animation etc.

MULTITHREADING

Contd.



Multithreading On a Dual Processor Desktop System

MULTITHREADING Contd.

ADVANTAGE:

- It doesn't block the user
- can perform many operations together so it saves time.
- Threads are **independent** so it doesn't affect other threads

CREATING THREAD

- Threads are implemented in the form of objects.
- The run() and start() are two inbuilt methods

which helps to thread implementation

- The **run()** method is the heart and soul of any thread
  - It makes up the entire body of a thread
- The run() method can be initiating with the help of **start()** method.

CREATING THREAD Contd.

1. By extending Thread class

2. By implementing Runnable interface

CREATING THREAD Contd.

1. By Extending Thread class

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

```
t1.start(); // run() method called through start() }
}
```

Output: thread is running…

CREATING THREAD Contd.

2. **By implementing Runnable interface**
   - Define a class that implements Runnable interface.
   - The Runnable interface has only one method, run(), that is to be defined in the method with the code to be executed by the thread.

CREATING THREAD Contd.

2. By implementing Runnable interface

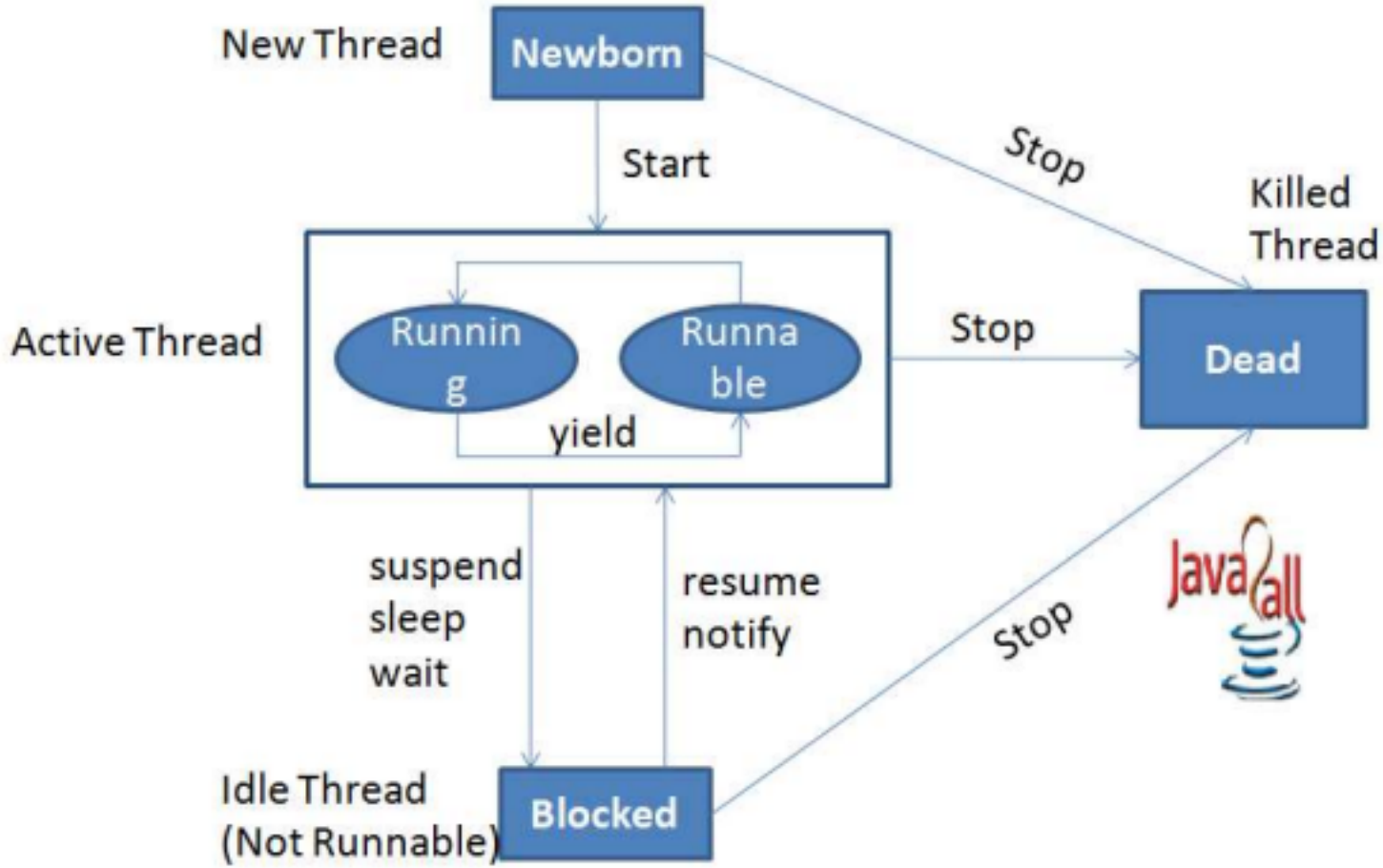
```
class Multi3 implements Runnable // Implementing Runnable interface {
public void run()
{
System.out.println("thread is running...");
}
public static void main(String args[])
{
Multi3 m1=new Multi3(); // object initiated for class Thread t1 =new
Thread(m1); // object initiated for thread t1.start();
 } }
```

Output: thread is running…

LIFE cycle of a thread

LIFE cycle of a thread contd.

- During the life time of a thread, there are many states it can enter.
- They include:
    1. Newborn state
    2. Runnable state
    3. Running state
    4. Blocked state
    5. Dead state



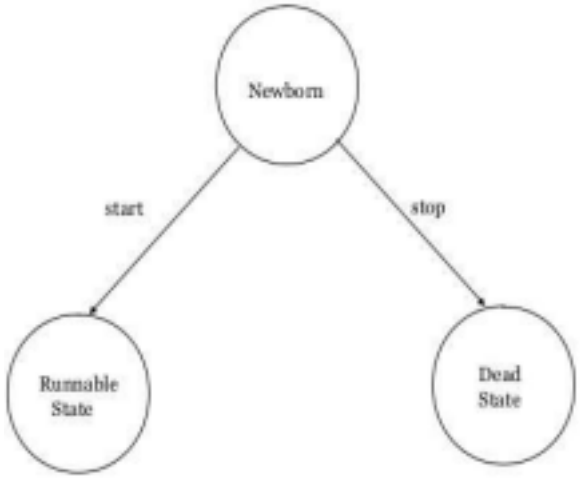
LIFE cycle of a thread contd.

Newborn State:

▪ The thread is born and is said to be in newborn state.

▪ The thread is not yet scheduled for running. ▪ At this state, we can do only one of the following:

• Schedule it for running using start() method.

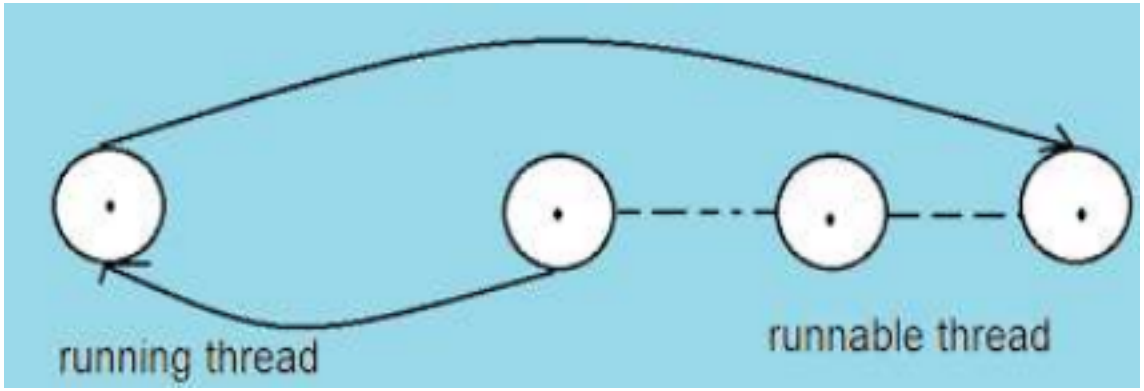


• Kill it using stop() method.

LIFE cycle of a thread contd.

Runnable State*:*

- The thread is ready for execution
- Waiting for the availability of the processor.
- The thread has joined the queue



LIFE cycle of a thread contd.

Running State:

- Thread is executing
- The processor has given its time to the thread for its execution.
- The thread runs until it gives up control on its own or taken over by other threads.

LIFE cycle of a thread contd.

Blocked State:

- A thread is said to be blocked
- It is prevented to entering into the runnable and the

running state.

- This happens when the thread is suspended, sleeping, or waiting in order to satisfy certain requirements.
- A blocked thread is considered "not runnable" but not dead and therefore fully qualified to run again.
- This state is achieved when we
  Invoke suspend() or sleep() or wait() methods.

LIFE cycle of a thread contd.

Dead State:

- Every thread has a life cycle.
- A running thread ends its life when it has completed executing its run( ) method. It is a natural death.

- A thread can be killed in born, or in running, or even in "not runnable" (blocked) condition.
- It is called premature death.
- This state is achieved when we invoke stop() method or the thread completes it execution.

Thread methods

- Thread is a class found in java.lang package.

Retrieves the name of running thread in the current context in String format

This method will start a new thread of execution by calling run() method of Thread/runnable object.

This method is the entry point of the thread. Execution of thread starts from this method.

This method suspend the thread for mentioned time duration in argument (sleeptime in ms)

By invoking this method the current thread pause its execution temporarily and allow other threads to execute.

This method used to queue up a thread in execution.

Once called on thread, current thread will wait till calling thread completes its execution

This method will check if thread is alive or dead

Stopping and blocking

Stopping a thread:

- To stop a thread from running further, we may do so by calling its ***stop()*** method.
- This causes a thread to stop immediately and move it to its dead state.
- It forces the thread to stop abruptly before its completion
- It causes premature death.
- To stop a thread we use the following syntax:

thread.stop();

Stopping and blocking

Blocking a Thread:

- A thread can also be temporarily suspended or blocked from entering into the runnable and subsequently running state,

    1. sleep(t) // blocked for ‘t’ milliseconds 2. suspend() // blocked until resume() method is invoked 3. wait() // blocked until notify () is invoked

Thread priority

- Each thread is assigned a priority, which affects the order in which it is scheduled for running.

- Java permits us to set the priority of a thread using the setPriority() method as follows:

ThreadName.setPriority(int Number);

Thread priority contd.

- The intNumber is an integer value to which the thread's priority is set. The Thread class defines several priority constants:

    1. public static int MIN_PRIORITY = 1
    2. public static int NORM_PRIORITY = 5
    3. public static int MAX_PRIORITY = 10

- The default setting is NORM_PRIORITY. Most user level processes should use NORM_PRIORITY.

Java synchronization

- Generally threads use their own data and methods provided inside their run() methods.
- But if we wish to use data and methods outside the thread's run() method, they may compete for the same resources and may lead to serious problems.
- Java enables us to overcome this problem using

a technique known as **Synchronization.**

For ex.: One thread may try to read a record from a file while another is still writing to the same file.

Java synchronization contd.

- When the method declared as synchronized, Java creates a "monitor" and hands it over to the thread that calls the method first time.

```
synchronized (lock-object)
{
```

..... // code here is synchronized

}

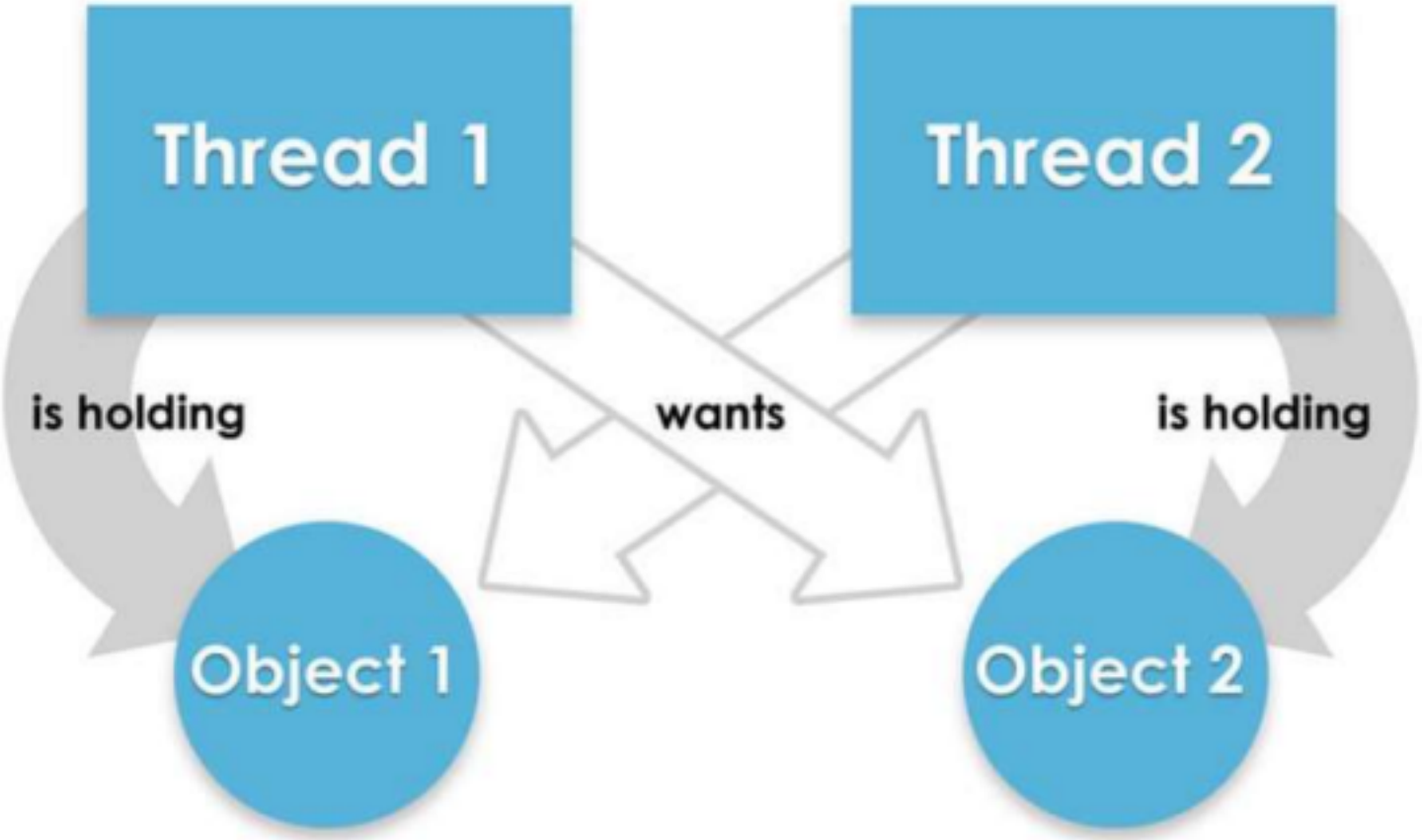
deadlock

- *Deadlock* describes a situation where two or more threads are blocked forever, waiting for each other.
- when two or more threads are waiting to gain control on a resource.

For example, assume that the thread A must access Method1 before it can release Method2, but the thread B cannot release Method1 until it gets

holds of Method2.

deadlock



THANK YOU…