Threads and Multithreading.

Introduction to Thread

- Thread is the basic unit of java program execution.
- Allows a program to operate more efficiently by doing multiple things simultaneously.
- It can be run without interrupting the main program running in the background.

Multithreads

- Multithreading in java is a process of executing multiple threads simultaneously.
- Lightweight, sub-process, and the smallest unit of processing.
- Java Multithreading is mostly used in games, animation, etc.

Advantages of Multithreading

- Doesn’t block the user.
- Saves time.
- Independent and doesn’t effect other threads while running.

Constructors in thread

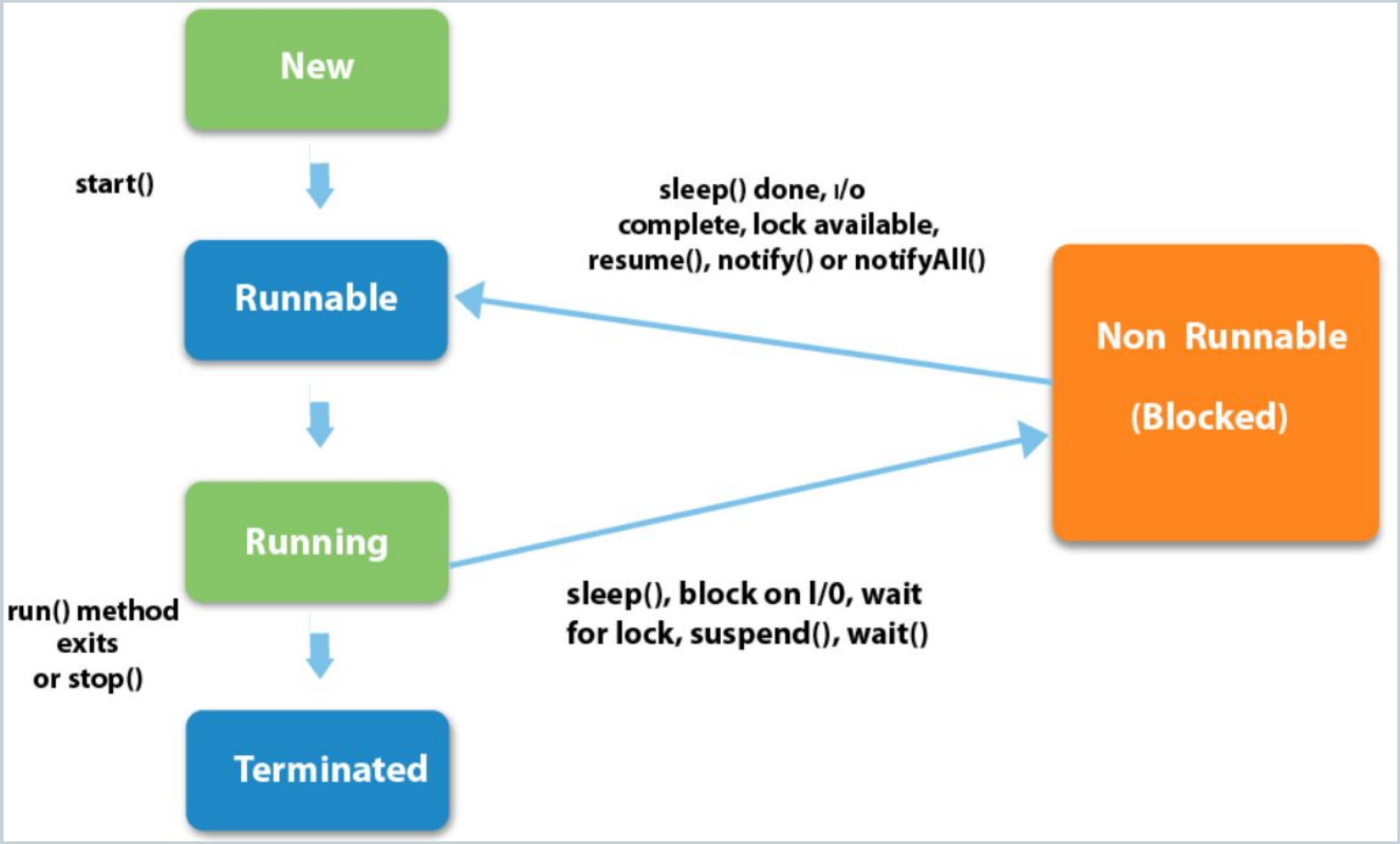
- Thread()
- Thread(String name)
- Thread(runnable r)
- Thread(runnable r, string name)

Life cycle of Thread

- It consist of 5 stages:

1. New
2. Runnable
3. Running
4. Non-runnable(Blocked)
5. Terminated.

Diagram


New
start()
Runnable
Running
run() method
exits
or stop()
Terminated
sleep() done, I/o
complete, lock available,
resume(), notify() or notifyAll()
Non Runnable
(Blocked)
sleep(), block on I/0, wait
for lock, suspend(), wait()

Creating a Thread

- Two methods:
- 1. By implementing runnable interface.
- 2. By extending thread class.

Creating a Thread

Syntax:

```
class classname extends Thread
{
  -------
  -------
     public void run()
     {
          //statements
     }
}
```

Creating a Thread

Syntax:

```
class classname implements Runnable
{
    -------
    -------
    public void run()
    {
        //statements
    }
}
```

Methods

- The run() and start() methods is used to implement thread.
- Run() method is the heart of any thread while implementation.
- The start() method helps to initiate the run() method.

Program 1

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...

Program 2

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...

Methods

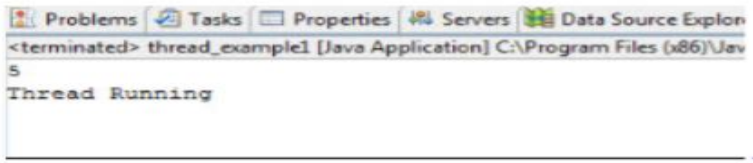
- run()
- Start()
- Sleep()
- getpriority()
- getName()
- getId()
- setName(string name)

Method s

Example code:

```
1.  package demotest;
2.  public class thread_example1 implements Runnable {
3.      @Override
4.      public void run() {
5.      }
6.      public static void main(String[] args) {
7.          Thread guruthread1 = new Thread();
8.          guruthread1.start();
9.          try {
10.             guruthread1.sleep(1000);
11.         } catch (InterruptedException e) {
12.             // TODO Auto-generated catch block
13.             e.printStackTrace();
14.         }
15.         guruthread1.setPriority(1);
16.         int gurupriority = guruthread1.getPriority();
17.         System.out.println(gurupriority);
18.         System.out.println("Thread Running");
19.     }
20. }
21.
```

Output :



Service thread

- Service thread are typically contain never ending loop for receiving and handling request.
- Such service threads can be convert to daemon threads.
- This daemon thread runs continuously in the background.
- Daemon thread in java is a service provider thread that provides services to the user thread
- Its life depend on the mercy of user threads i.e. when all the user threads dies, JVM terminates

Creating Service thread

- By setDaemon() method- To specify that it is a daemon thread.
- By isDaemon() method- To check that current is daemon.

Important points for Daemon thread

Simple example of Daemon thread in java

```
public class TestDaemonThread1 extends Thread{
 public void run(){
  if(Thread.currentThread().isDaemon()){//checking for daemon thread
   System.out.println("daemon thread work");
  }
  else{
  System.out.println("user thread work");
 }
 }
 public static void main(String[] args){
  TestDaemonThread1 t1=new TestDaemonThread1();//creating thread
  TestDaemonThread1 t2=new TestDaemonThread1();
  TestDaemonThread1 t3=new TestDaemonThread1();

  t1.setDaemon(true);//now t1 is daemon thread

  t1.start();//starting threads
  t2.start();
  t3.start();
 }
}
```

Output:

```
daemon thread work
user thread work
user thread work
```

Schedule task using JVM

- To schedule the thread to run later.
- For eg: you can check the value of a variable that many threads are updating after every 10 seconds.

Class to schedule task using thread

- 1- TimerTask class: a task that can be scheduled for one-time or repeated execution by a timer.

- 2- Timer class: a facility for thread to schedule tasks for future execution in a background thread.

CONT…

- TimerTask Class-
- 1. Constructor:
- TimerTask()- creates a new timer task.

- 2. Methods:
- void cancel()
- void run()
- Long scheduledExecutionTime()

Cont…

- Timer Class-
- 1. Constructor
- Timer()

- 2. Methods
- Void cancel()
- Void schedule(TimerTask task, Date time)
- Void schedule(TimerTask task, Date firsttime, long period)

Thread Safe

- By doing Thread safe, it removes the danger of variables of their values being changed asynchronously by threads other than the current one.
- It belongs to one thread alone, different threads have separate copies.

1. ThreadLocal class

- This class provides ThreadLocal variables.
- Thread local objects are private, static variables in classes that wish to associate state with a thread(Eg: User ID)
- It is declared as private and static.
- Eg:

private ThreadLocal threadLocal = **new** ThreadLocal();

Constructors:

1. ThreadLocal()

Methods:

1. Object get()
2. Object initialValue()
3. void set(Object value)

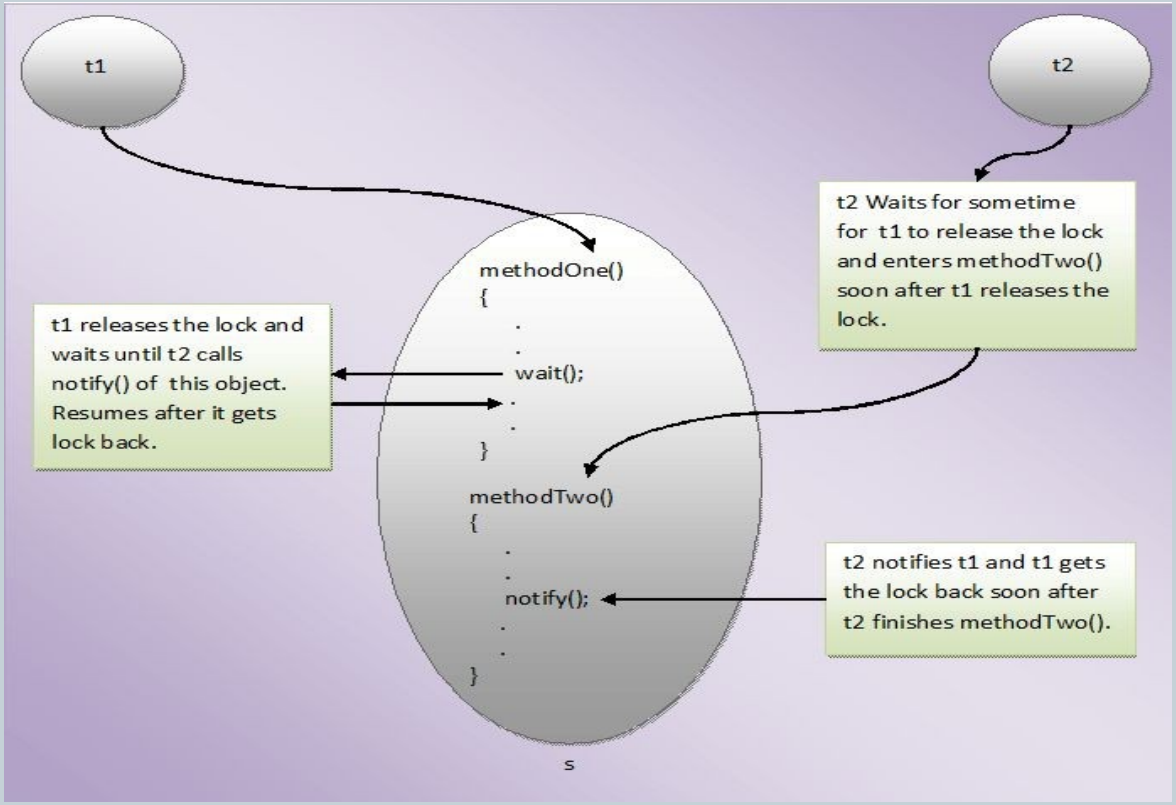
Inter thread Communication

- In multithreading programming, there's need of some communication between threads.
- It helps thread to notify if certain conditions change.
- Eg: Producer consumer problem.

Methods

- notify()- Wakes up a single thread that is waiting on this object.
- notifyAll()- Wakes up all threads that are waiting on this object.
- wait()- Causes current thread to release the lock and wait until either another thread invokes the notify() method.

Cont…


t1
t2
t2 Waits for sometime for t1 to release the lock and enters methodTwo() soon after t1 releases the lock.
methodOne()
{
.
.
wait();
.
.
}
t1 releases the lock and waits until t2 calls notify() of this object. Resumes after it gets lock back.
methodTwo()
{
.
.
notify();
.
.
}
t2 notifies t1 and t1 gets the lock back soon after t2 finishes methodTwo().
s

Synchronizing Threads

- Synchronization is the way to avoid data corruption caused by simultaneous access to the same data by multiple threads.
- For example, if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

Cont…

- Syntax: synchronized(objectidentifier) { }
  here objectidentifier is a reference to the object.

Example code

```
1.  //example of java synchronized method
2.  class Table{
3.   synchronized void printTable(int n){//synchronized method
4.     for(int i=1;i<=5;i++){
5.       System.out.println(n*i);
6.       try{
7.        Thread.sleep(400);
8.       }catch(Exception e){System.out.println(e);}
9.     }
10.
```

Example code

```
11.  }
12. }
13.
14. class MyThread1 extends Thread{
15. Table t;
16. MyThread1(Table t){
17. this.t=t;
18. }
19. public void run(){
20. t.printTable(5);
21. }
22.
23. }
24. class MyThread2 extends Thread{
25. Table t;
26. MyThread2(Table t){
27. this.t=t;
28. }
29. public void run(){
30. t.printTable(100);
31. }
32. }
```

Example code

```
33.
34. public class TestSynchronization2{
35. public static void main(String args[]){
36. Table obj = new Table();//only one object
37. MyThread1 t1=new MyThread1(obj);
38. MyThread2 t2=new MyThread2(obj);
39. t1.start();
40. t2.start();
41. }
42. }
```