Java Programming

Follow:

Blog: www.rkkeynotes.blogspot.com

YouTube: RK Keynotes

Instagram: RK KEYNOTES (@rk_keynotes)

LinkedIn: https://www.linkedin.com/in/ravikumarrn/

Facebook: https://www.facebook.com/Ravikumar.R.natarajan

Regards,

Ravikumar R N

Unit - 4 Multithreading and Generic Programming

1. What is Thread?

- Thread is a tiny program running continuously. It is called as light weight process. Any single path of execution is called thread. | A thread is also called flow of execution.
- Switching between threads automatically done by JVM

1.1.Concurrent Programming

- It means parallel execution of processes.
- Two basic units of execution,
    - **Processes** – it has self contained execution environment. Has its own memory space.
    - **Threads** – Threads exist within process – every process has at least one. Threads share process's resources including memory and open files.

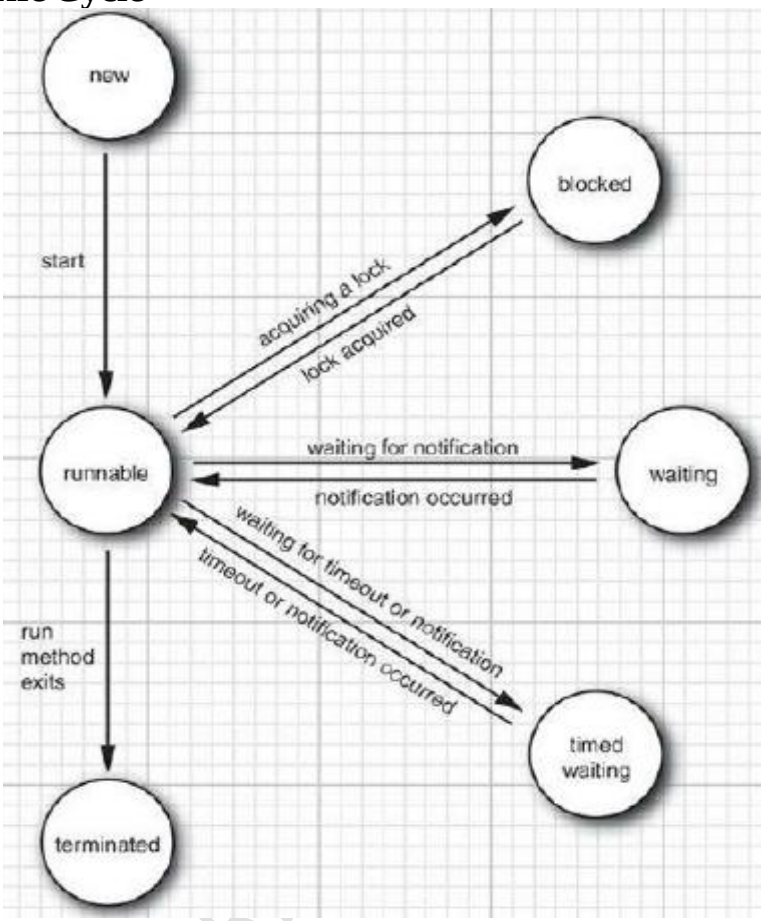
A computer system normally has many active processes and threads. Processing time for a single core is shared among processes and threads through an OS feature called **time slicing**.

S.No	Thread	Process
1.	It is a light-weight process	It is a heavy-weight process
2	Threads do not require separate address space. It runs in the address space of process to which it belong to.	Each process requires separate address space to execute.

1.2. Difference between Multithreading and Multitasking

S.No	Multithreading	Multitasking
1.	Thread is a fundamental unit of multithreading.	Process is a fundamental unit of multiprocessing environment.
2.	Multiple parts of a single program gets executed in multithreading envi.	Multiple programs get executed in multiprocessing envi.
3.	During multithreading the processor switches b/w multiple threads of the program.	During multiprocessing the processor switches b/w multiple programs.
4.	It is cost effective bcz cpu can be shared among multiple threads at a time.	It is expensive because when a particular process uses cpu other processes has to wait.
5.	Highly efficient	Less efficient
6.	It helps in developing application programs.	It helps in developing OS programs.

2. Thread Life Cycle



1. When one creates a new Thread object, it is created with the resources it needs to run. But when it is instantiated it is created in blocked state.
2. A blocked thread cannot run until the block is released.
3. Calling a thread **start()** method removes the initial block and causes the scheduler to call the **run()** method.
4. This places the thread in the queue for CPU time.

New state

- When a thread starts its life cycle it enters in the new state or a create state.

Runnable state

- This is a state in which a thread starts executing. The **scheduler (a part of JVM)** selects a thread from runnable pool.

Blocked state

- When a particular thread issues an input/output request then OS sends it to blocked state until I/O gets completed. After I/O completion the thread sent back to runnable state.

Waiting state

- Sometimes one thread has to wait because another thread starts executing.

Timed waiting

- This state is to keep a particular thread waiting for some time interval. This allows to execute high prioritized threads first. After waiting, the thread enters in runnable state.

Terminated state

- After successful completion of the execution the thread in runnable state enters the terminated state.

3.Creating Threads

Two approaches,

1. Using **Thread** class
2. Using **Runnable** interface

The **run()** method is the most important method in any thread programming. Using this method thread behavior can be implemented.

Syntax:

```
public void run()
{
        //statements
}
```

Methods used in Threads

getName() – obtain thread name

getPriority() – obtain a thread priority

isAlive() – check if a thread is still running

join() – wait for thread to terminate

run() – entry point of thread

sleep() – suspend a thread for a period of time

start() – start a thread by calling run method

Other methods

Destroy() or stop() – to terminate execution of thread

Running thread calls Wait() then Notify() and NotifyAll() message sent

Suspend() or yield() – to wait

Interrupted() – thread instance will set interrupted flag

What is the use of flag in programming?

It is a variable. A **flag** is a value that acts as a signal for a function or process. The value of the **flag** is used to determine the next step of a **program**. **Flags** are often binary **flags**, which contain a boolean value (true or false).

1.Example using Thread class

```
class mythread extends Thread
{
        public void run()
        {
        System.out.println("Thread is created!!");
        }

        public static void main(String args[])
        {
        mythread t= new mythread();
        t.start();
        }
}
```

2.Example using Runnable interface

```
class runnableDemo implements Runnable{
  public void run()
{
   System.out.println("My thread is in running state.");
  }
  public static void main(String args[])
{
    runnableDemo obj=new runnableDemo ();
    Thread t1 =new Thread(obj);
    t1.start();
 }
}
```

Creating Multiple Threads Using Thread class:

```
class A extends Thread
{
        public void run()
        {
        for(int i=0; i<=5; i++)
        {
        System.out.println(i);
}}}

class B extends Thread
{
        public void run()
        {
        for(int i=10; i>=5; i--)
        {
        System.out.println(i);
}}}
class multithread
{
public static void main(String args[])
{
A t1 = new A();
B t2 = new B();
t1.start();
t2.start();
}}
```

//**Task** Create the above using Runnable interface

4. Synchronized Thread

Java Synchronization allows only one thread to access the shared resource. This will overcome problems like i) Thread interference and ii) Memory consistency errors. iii) Inconsistency problems.

The synchronization concept is based on monitor, which is lock and unlock. When a thread owns this monitor other thread cannot access the resources. Other threads will be in waiting state.

Two ways to achieve synchronization,

1) Using synchronized methods
2) Using synchronized blocks or statements

Rules:

1. Constructors, Classes and variables cannot be synchronized
2. Each object has one lock
3. A thread can acquire more than one lock.
4. When synchronization applied it is called critical section (one thread process can access resource at a time).

1.Example for non-synchronized and synchronized methods:

```
class Table{
        void printTable(int n)//method not synchronized  // synchronized void printTable(int n)
                {
                 for(int i=1;i<=5;i++){
                        System.out.println(n*i);
                 try{
                         Thread.sleep(400);
                  }catch(Exception e){System.out.println(e);}
                                        }
                }
           }

   class MyThread1 extends Thread{
          Table t;
          MyThread1(Table t)
          {          this.t=t;
          }
          public void run()
          {         t.printTable(5);
          }
        }
   class MyThread2 extends Thread{
            Table t;
          MyThread2(Table t){
          this.t=t;
          }
   public void run(){
   t.printTable(100);
          }
   }

   class TestWOSync{
          public static void main(String args[]){
          Table obj = new Table();//only one object
          MyThread1 t1=new MyThread1(obj);
          MyThread2 t2=new MyThread2(obj);
          t1.start();
          t2.start();
```


Non-Sync O/P

```
5
100
10
200
15
300
400
20
25
500
```

Sync O/P

```
5
10
15
20
25
100
200
300
400
500
```

```
}
}
```

What is monitor lock in Java threads?

The Java programming language provides multiple mechanisms for communicating between threads. The most basic of these methods is synchronization, which is implemented using monitors. Each object in Java is associated with a monitor, which a thread can **lock** or **unlock**.

What is intrinsic lock in Java?

intrinsic lock: an "instance **lock**", attached to a single object.
A "static **lock**", attached to a class.

2. Example for synchronized Block: (synchronized statements must specify the object that provides the intrinsic lock**)**

```
class Table{
        void printTable(int n)
                {
                Synchronized(this) //Sync.Block
                {
                        for(int i=1;i<=5;i++){
                                System.out.println(n*i);
                        try{
                                Thread.sleep(400);
                            }catch(Exception e){System.out.println(e);}
                }                               }
                }
           }

  class MyThread1 extends Thread{
         Table t;
         MyThread1(Table t)
        {          this.t=t;
         }
         public void run()
        {         t.printTable(5);
        }
       }
  class MyThread2 extends Thread{
          Table t;
        MyThread2(Table t){
         this.t=t;
         }
  public void run(){
  t.printTable(100);
         }
  }
```

```
class TestWOSync{
    public static void main(String args[]){
    Table obj = new Table();//only one object
    MyThread1 t1=new MyThread1(obj);
    MyThread2 t2=new MyThread2(obj);
     t1.start();
     t2.start();
}
}
```

5. Inter-thread Communication

Definition: Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Two or more threads communication with each other by exchanging the messages. This mechanism is called inter-thread communication.

Polling is a mechanism implemented in a loop in which certain condition is repeatedly checked. Ex: producer – consumer problem. Both must work in coordination to avoid wastage of CPU cycles.

In this, producer has to wait for the consumer to finish consuming data. Similarly, consumer has to wait for the producer to produce data. In polling system either consumer or producer will waste CPU cycles while waiting. To avoid polling, the following in-built methods used,

- wait() – waits until other thread calls notify or notify all
- notify() – if a particular thread is in sleep mode it can be resumed using notify() call.
- notifyAll() – this method resumes all the threads which are in sleep mode.

Example for Inter-Thread communication

```
class Customer
{
    int amount=10000;
    synchronized void withdraw(int amount)
    {
        System.out.println("Going to withdraw...");

        if(this.amount<amount)
        {
        System.out.println("Less balance; waiting for deposit...");
        try{wait();}catch(Exception e){}
        }
    this.amount-=amount;
    System.out.println("Withdraw of " +amount+" completed...");
    }

    synchronized void deposit(int amount)
    {
        System.out.println("going to deposit...");
        this.amount+=amount;
        System.out.println("Deposit of " + amount+ " completed... ");
```

```
                notify();
        }
}

class InterThread
{
        public static void main(String args[])
        {
                final Customer c=new Customer();
                new Thread(){
                        public void run(){c.withdraw(500);}
                }.start();

                new Thread(){
                        public void run(){c.deposit(12000);}
                }.start();
        }
}
```

Output: Going to withdraw...
Less balance; waiting for deposit...
Going to deposit...
Deposit of 12000 completed...
Withdraw of 500 completed

Note: suspend(), resume(), stop() all are deprecated by java 2. Because suspend will cause serious system failures. If one method suspended all will get affected.
Resume method cannot be used with suspend. If a method is stopped it will enter into corrupted state.

6. Daemon Threads

- **Daemon thread in java** is a service provider thread that provides services to the user thread.
- It is a low priority thread which runs in the background.
- Ex: Garbage Collector

Two types of threads,

- **setDaemon**(true/false) – to set a daemon thread.
- public Boolean **isDaemon** – to determine the thread is daemon or not

Why JVM terminates the daemon thread if there is no user thread?

- The purpose of the daemon thread is that it provides services to user thread for background supporting task. If there is no user thread, why should JVM keep running this thread. That is why JVM terminates the daemon thread if there is no user thread.

Example for Daemon Thread:

```
public class DemoDem 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)
{
        DemoDem t1=new DemoDem ();//creating thread
        DemoDem t2=new DemoDem ();
        DemoDem t3=new DemoDem ();

 t1.setDaemon(true);//now t1 is daemon thread
 t1.start();//starting threads  //if t1 is started before setDemon it throws illegalstateException
 t2.start();
 t3.start();
 }
 }
```

Output: daemon thread work
user thread work
user thread work

7.Thread Groups

Definition: Threads groups provide a mechanism for collecting multiple threads into a single object and manipulating those threads all at once.

Ex: start, resume, stop all the threads with in a group.

Java thread group is implemented by ***java.lang.ThreadGroup*** class.

Two constructors,

- ThreadGroup(String name)
- ThreadGroup(ThreadGroup parent, String name)

Methods of ThreadGroup Class

- int activeCount() – returns no.of threads running in current group
- int activeGroupCount() – returns no.of active groups
- void destroy() – destroys thread group and sub groups
- string getName() – returns name of the group
- void interrupt() – interrupt all threads
- void list() – print info.of the group to standard console
- ThreadGroup getParent() – returns the parent of the group

Example for ThreadGroup:

```
public class ThreadGroupDemo implements Runnable
{
        public void run()
        {
```

```
                System.out.println(Thread.currentThread().getName());
        }
         public static void main(String[] args)
        {
                ThreadGroupDemo runnable = new ThreadGroupDemo();

                ThreadGroup tg1 = new ThreadGroup("Group A");
                ThreadGroup tg2 = new ThreadGroup("Group B");

                System.out.println("Thread Group Name: "+tg1.getName());
                Thread t1 = new Thread(tg1, runnable,"One");
                Thread t2 = new Thread(tg1, runnable,"Two");

                t1.start();
                t2.start();

                System.out.println("Thread Group Name: "+tg2.getName());
                Thread t3 = new Thread(tg2, runnable,"Three");
                //Thread t4 = new Thread(tg2, runnable,"Four");


                t3.start();
                //t4.start();

                System.out.println("The active threads are: "+tg2.activeCount());
                tg1.list();

     }
    }
```

Output:

```
D:\java\Unit_4\threadGroup>java tgtry
Thread Group Name: Group A
Thread Group Name: Group B
One
Three
The active threads are: 1
Two
java.lang.ThreadGroup[name=Group A,maxpri=10]
    Thread[Two,5,Group A]
```

8.Generic Programming

Generic was first introduced in java 5. Generic programming enables the programmer to create classes, interfaces and methods in which **type of data is specified as a parameter**.

Advantages:

1. No need to create separate methods to handle different data type.
2. Allows code reusability
3. Provides type safety
4. Individual Type Casting is not needed

8.1 Type Parameters

Naming conventions are important to learn generics. Commonly used parameters are,

- T – Type
- E – Element
- K – Key
- N – Number
- V – Value

8.2 Generic Classes

A generic class contains one or more variables of generic data type.

Example:

```
class gen<T> //brackets indicates the class is of generic type // class Test<T, U>
{
        T obj; //an object of type T is created
        gen(T obj)
        {
                this.obj =obj;
        }
        public void print()
        {
                System.out.println(obj);
        }
}
class Demogen
{
        public static void main(String args[])
        {
                gen<Integer> iob=new gen<Integer>(15);
                iob.print();

                gen<String> sob=new gen<String>("Hello");
```

```
//multiple parameters
Class Test<T, U>
{
        T obj1;
        U obj2;
        Test(T obj1, U obj2)
        {
        This.obj1 = obj1;
        This.obj2 = obj2;
        }
        Public void print()
        {System.out.println(obj1);
        System.out.println(obj2);
        }}
Class main
{   Psvm(String[] args)
   {
     Test <String, Integer> obj = new
Test <String, Integer> (“ABC”, 15);
     Obj.print();
   }}
```

```
                sob.print();
        }
}
```
Output: 15
Hello

8.3 Generic Methods

Sometimes we don't want whole class to be parameterized, in that case java generic method created.

Generic functions that can be called with different types of arguments based on the type of arguments passed to generic method, the compiler handles each method.

```
// A Simple Java program to show working of user defined
// Generic functions

class Test
{
   // A Generic method example
   static <T> void genericDisplay (T element)
   {
      System.out.println(element.getClass().getName() +
                  " = " + element);
   }

   public static void main(String[] args)
   {
       // Calling generic method with Integer argument
      genericDisplay(11);

      // Calling generic method with String argument
      genericDisplay("Hi");

      // Calling generic method with double argument
      genericDisplay(1.0);
   }
}
```
Output :
java.lang.Integer = 11
java.lang.String = Hi
java.lang.Double = 1.0

8.4 Bounded Types

Bounded type is used to limit or restrict the type of object passed to the parameterized type. Using bounded types, we can make the objects of generic class to have data of specific derived types.

Ex: if we want a generic class to work only with numbers

Syntax: <T extends SuperClass> i.e. <T extends Number>

```
class Demo<T extends Number>
{
        T t;
        public Test(T t)
        {
                this.t =T;
        }
        public T getT()
        {
                return t;
        }
}
public class BoundDemo
{
public static void main(String[] args)
{
Demo<Number> obj1 = new Demo<Number>(123);
System.out.println("The integer is:"+obj1.getT());

//Demo<String> obj1 = new Demo<String>("Hello"); //compile time error
//System.out.println("The integer is:"+obj1.getT());  }}
```


8.5 Restrictions and Limitations

1. Primitive type parameters not allowed. Ex: stack<int>
2. Instantiation of generic parameters T is not allowed. Ex: new T()
3. Arrays of parameterized type not allowed.
4. Static fields & static methods with type parameters are not allowed.
5. Cannot use cast
6. Cannot overload

2 Mark

1. What is Thread?
2. What is Concurrent Programming?
3. **D/B Thread and Process?**
4. **Difference between Multithreading and Multitasking?**
5. What is the use of flag in programming?
6. What is intrinsic lock in Java?
7. What is monitor lock in Java threads?
8. What is Inter-thread Communication?
9. What is Polling?
10. **What is Daemon Threads? List its types?**
11. Why JVM terminates the daemon thread if there is no user thread?
12. What do you mean by Thread Groups?
13. What is Generic Programming?
14. **List Restrictions and Limitations in Generic Programming.**

13 marks

1. **Explain Thread Life Cycle with diagram.**
2. Explain Creating Threads with example.
3. Write a program to Create Multiple Threads Using Thread class.
4. **Explain Synchronized Thread using methods and blocks.**
5. **Write a program to show Inter-thread Communication process.**
6. Explain Daemon Threads types with example.
7. Explain Thread Groups with example.
8. **Write a program using Generic Class and Methods.**

Task

1. **Write a java program that prints the numbers from 1 to 10 line by line after every 10 seconds.**