16/02/019

Unit-IV :- multithreading

Crazy notes Jntuk

⇒ Thread :-

→ A thread represents a seperate path of execution of a group of statements.

→ in Java program, if we write a group of statements then these statements are excuted by JVM one by one.

→ This execution is known as thread. that means in every Java programming there is always a thread is running internally.

→ This thread is used by Java virtual machine to execute the program statement.

→ In a thread the way the statements are executed can be defining two ways :-

① Single tasking :-

A task means doing some calculations (or) processing in a single tasking environment only one task is given to the processor at a time this means we are westing a lot of processor time and the microprocessor has to sit ideal without any job. for a long time.

→ This is the drawback in single tasking.

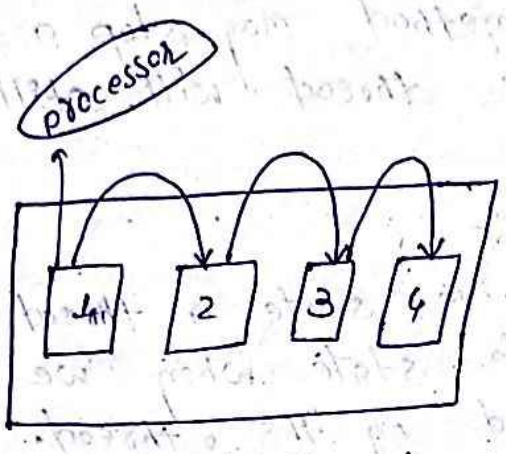
② multitasking :-

→ in multitasking we can give several task at a time. for example there are four task then we want to execute,

first we load them into the memory the memory is divided into four parts and task are loaded in the memory.

Now, the microprocessor has to execute these task by ~~sett~~ setting the time to execute these task at a time.

→ This can be represented as—



18/02/09

⇒ Creating a simple thread :-

eg:

```
class current
{
Public static void main (string args[])
{
System.out.println ("current thread");
Thread t = Thread.currentThread();
System.out.println("current thread:"+t);
System.out.println("Its name:"+t.getname);
}
}
```

o/p:- current thread
current thread: main 5;
main

⇒ Thread Life cycle:-

→ The process of starting from the birth of a thread until it's deaths is known as thread life cycle.

→ The states in thread life cycle are:-

(i) Born() state:- A thread will be born when it is created by the user with the help of following statements.

thread e = new thread();

(ii) start:- After the creation of a thread is done in the born() state the thread goes into the

runnable state when we applied the start method

③ yield():- This method may stop a thread temporarily but the thread will still in runnable state.

→ sleep or wait():-

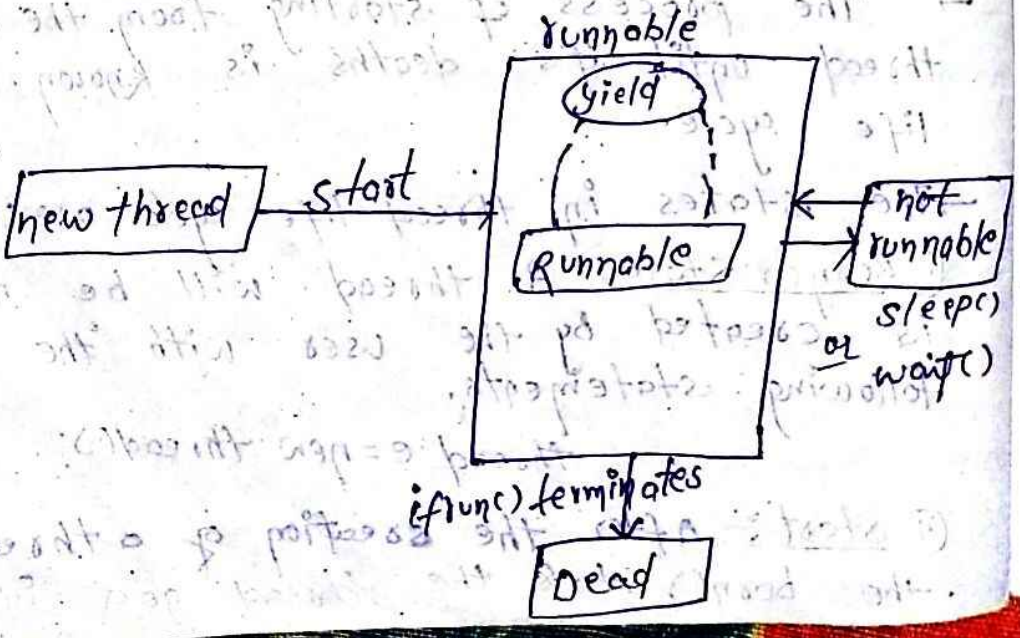
from a runnable state a thread may get into a not runnable state when we apply sleep or wait method on the thread.

Run():- A thread is terminate from the memory only if comes out of Run method.

→ A thread is created in the born state and it is execute by the start method After that the thread entered into the runnable state when we apply the sleep (or) wait method on the thread it goes into not runnable state from not runnable state the thread comeback to the runnable state and continues running the statement.

→ The thread dies when it come out of run method This state transaction is known as thread life cycle.

→ This can be represented as-



19/07/18

⇒ Creating a Thread :—

In a java program a default thread that is ~~not~~ known as main is already available.

apart from this main thread we can also create our home thread in a program. The user can create his own thread in two ways—

① by using the extends keyword

② by implementing a runnable interface.

① By using the extends keyword—

We can create a thread class by ~~of~~ using extence keyword. This can be represented as—

```
class myclass extends thread
```

in the above statement the thread after the extends keyword is a free defined thread class.

② By implementing a runnable interface—

By using the runnable interface it can represent a thread class as—

```
class myclass implements runnable
```

in the above statement runnable is a free define interface.

→ After creating a thread class we have to write a run method for representing the code of a thread. This can be represented as—

```
Public void run()
{
  statements; (or) code of the thread;
}
```

→ After the run method we have to create an object to myclass. This can be represented as—

```
myclass obj=new myclass();
```

→ After creating an object to my class we have to create a thread and attach the thread to the object. This can be represented as—

```
Thread t=new Thread(obj);
```
(or)
```
Thread t=new Thread(obj,"Thread name");
```

→ At last we have to run the thread by using the start method. This can be represented as—

```
t.start();
```

eg:-

```
class myThread extends Thread
{
 Public void run()
 {
  for(int i=1; i<=10000; i++)
  {
   System.out.println(i);
  }
 }
}
class demo
{
 Public static void main(String args[])
 {
  mythread obj=new mythread();
  Thread t=new Thread(obj);
        t.start();
 }
}
```

30/02/2019

⇒ Terminating a Thread normally :-

if we want to stop a running thread normally we have to follow some rules:-

→ To terminate a thread abnormally the following code can be use-

```
import java.io.*;
class myThread extends Thread
{
  boolean stop=False;
  Public void run()
  {
   for(int i=1 ; i<=10,000; i++)
   {
    System.out.println(i);
    if(stop)
    return;
   }
  }
}
class demo
{
 Public static void main(string args[]) throws exception
 {
  myThread obj=new myThread();
  Thread t=new Thread(obj);
  System.out.println t.start();
  System.in.read();
  obj.stop=true;
 }
}
```

⇒ Thread priorities :-

When threads are created a thread schedulor program in JVM will load them into memory and execute them.

→ This schedulor will allow more JVM time to those threads which are having highest prioritys.

→ The priority of a thread will be ranged from 1 to 10. The minimum priority of a thread is 1, and the maximum priority of a thread is 10.

→ The default (or) the normal priority of a thread is 5.

→ The minimum priority of a thread can be found by using -

`Thread.MIN-PRIORITY`

→ The maximum priority of a thread can be found by using -

`Thread.MAX-PRIORITY`

→ The Normal priority of a thread can be found by using -

`Thread.NORM-PRIORITY`

→ when two task are assigned to two threads with different priorities as- 2 & 5 then the thread with higher priority numbers will give more JVM time and hence it will complete the task either then the thread priority ~~number~~ number is 2.

eg: class myclass extends Thread.

```
{
  int Count=0;
```

```
Public void run()
{
  for(int i=0; i<=10,000; i++)
  count++;
  System.out.println("completed Thread:"+ Thread.current
                                    Thread().getName());
  System.out.println("It's priority:"+ Thread.current
                                    Thread().getpriority());
  }
}
class prior
{
  Public static void main (string args[])
  {
    myclass obj=new myclass();
    Thread t1=new Thread (obj, "one");
    Thread t2=new Thread (obj, "Two");
    t1.setpriority(2);
    t2.setpriority(Thread.NORM_PRIORITY);
    t1.start();
    t2.start();
  }
}
```

op: completed thread: Two
it's priority: 5
completed thread: one
it's priority: 2

23/02/19

⇒ Synchronized Thread:-

when a thread is already acting an object and preventing another thread from acting on the same object is known as thread synchronization (or) thread safe. The object on which the threads are synchronized is known as synchronized object. we can synchonized an object in two ways-

① using synchronized block:-

In this we can combined a group of statements of the object with in a synchronized block. this can be represented as-

```
Synchronized (object)
{
  statements;
}
```

The above block represents an object to be blocked are synchronized and what even statements that is written in the synchronized block are available through only one thread at a time. They are not available to more than one thread simultaneously.

② using Synchronized keyword:-

we can synchonized an entire method by using synchronized keyword.

eg: if we want to synchonize a method then add the synchonized keyword before the method name. This can be represented as-

```
Synchonized void display()
{
  statements;
}
```

eg: class reverse implements Runnable

```
{
    int available = 1;
    int wanted;
    reversed(int i)
    {
     wanted = i;
    }
    Public void run()
    {
     Synchronized(this);
     {
      if(available >= wanted)
     {
     String name = Thread.currentThread().getName();
     System.out.println(wanted + "Before reverse for" + name);
     try
     {
     Thread.sleep(15000);
      available = available - wanted;
     }
     catch(Interrupt Exception ie)
     {
     }
    }
   else
   System.out.println("sooo no seats");
   }
   class safe
   {
   Public static void main(String args[])
   {
```

```
reverse obj = new reverse (1);
Thread t1 = new Thread (obj);
Thread t2 = new Thread (obj);
t1.setname ("first person");
t2.setname ("second person");
t1.start
t2.start
}
}
```

o/p:-

⇒ Thread Communication:-

→ In some cases two or more thread should communicate with each other for example a consumer thread is waiting for a producer to produce the data then the producer thread completes the production of the data then the consumer thread should take the data and use it.

→ In producer class we take a string buffer object to store the data. In this case it takes some numbers from 1 to 10. These numbers are added to string buffer object. After that we take another boolean variable as a data provider and ~~when~~ initialize it false. The purpose is to make this data provider available when the production of number is completed. producing the data is done by appending numbers to string buffer

by using a for loop.

→ when the producer is busy for producing the data then the consumer will check if the data provider is true (or) not.

→ if the data provider is true then consumer takes data from string buffer and uses it. if it is shows false then the consumer will see for some time and then again checks the data provider.

eg:

```
class producer extends Thread
{
  string Buffer sb;
  boolean data prodover=false;
  producer();
  {
    sb=new string Buffer();
  }
  Public void run()
  {
  for(int i=1; i<10; i++)
   {
    try
    {
    sb.append(i);
    Thread.sleep(100);
    System.out.println("appending");
    }
    catch(Exception e)
    {
    }
  }
  data prodover = True;
  }
}
```

```
class consumer extends Thread
{
  producer prod;
  consumer (producer prod)
  {
    this.prod = prod;
  }
  Public void run()
  {
   try
   {
   while(!prod.data prodover)
   {
    Thread.sleep(10);
   }
  }
  catch (Exception e)
  {
  }
  System.out.println(prod.s[illegible];
  }
}
class communicate
{
psvm (string args[]) throws IoException
{
producer obj1 = new producer();
consumer obj2 = new
Thread t1 = new Thread(obj1);
   "  t2 =  "    "  (obj2);
  t1.start();
  t2.start();
}}
```

Reading the data from the file

file

→ A collection of records can be known as file.

Stream

→ A stream represents the flow of data from one place to another place.

→ A stream can be classified into two types:

(i) input stream (ii) output stream.

input stream

→ which receive or read the data from the file.

output stream

→ which sends or write the data from the file.

→ for reading the data from the keyword we can use the following statements.

```
DataInputStream dis = new DataInputStream (System.in);
```

→ In the above statement we are attaching the keyword to DataInputStream obj the keyword is represented System.in, now the DataInputStream can read the data from the keyword.

Ex:

```
import java.io.*;
class createfile
{
public static void main (String args[]) throws IOException
{
DataInputStream dis = new DataInputStream(System.in);
FileInputStream fin = new FileInputStream ("myfile.txt");
```

```
System.out.println("enter text(@ at end):");
char ch;
while((ch=(char)dis.read()!='@'))
{
    fin.write(ch);
    fin.close();
}
}
```

Opl:-

⇒ writing the data to the file

Ex:-

```
import java.io.*;
class copy file
{
public static void main(String args[])throws IoException
{
FileInputStream in=null;
FileOutputStream out=null;
try
{
in=new FileInputStream("Input.txt");
out=new FileOutputStream("output.txt");
int c;
while((c=in.read())!=-1)
{
out.write(c);
}
}
```

```
catch (IOException e)
{
    System.out.println(e);
    System.exit(-1);
}
finally
{
    if (in != null)
    {
        in.close();
    }
    if (out != null)
    {
        out.close();
    }
}
```

Random access file

→ In random access files, we can access the information that store in the file directly without going through all the files.

→ A random access file has a file pointer that moves forward, when we read the data from the files.

→ The file pointer is nothing but a cursor, we can get the value of the file pointer by using getFilepointer() method.

→ When we create an object to random access file, the file pointer is said to zero(0), we can set the file pointer location at specific position in the file by using seek() method.

→ The length of a random access file can be formed by

```
Random Access file ref = new
Random Accessfile ("random test.text", "rw");
```

Ex:

```
import java.io.*;
class Randomfile
{
public static void main (String args[])
{
  try
    {
```

```
RandomAccessFile file = new RandomAccessFile("stddoc", "rw");
file.setLength(0);
for(int i=0; i<50; i++)
file.writeInt(i);
System.out.println("length of the file after writing data
                    is: " + file.length());
file.seek(0);
System.out.println("first number is: " + file.readInt());
file.seek(1*4);
System.out.println("second number is: " + file.readInt());
file.writeInt(101);
file.seek(file.length());
file.writeInt(50);
System.out.println("current length of file: " + file.length());
}
catch(Exception e)
{
 e.printStackTrace();
 }
}
}
```

Reading / writing using Random Access file:-

Ex:

```
import java.io.*;
class Random
{
Public static void main (String args[])
{
  Random file = null;
  try
  {
  file = new Random Access file ("Rand.dat");
  file.writeChar('x');
    " . "  Int (555);
    "  "  Double (3.1456);
  file.seek (0);
System.out.println ("file.read char ());
    "  "  ("file.read Int ()");
    "  "  ("file.read Double()");
System.out.println ("file.read Int());
  file.seek (file.length ());
    ".writeBoolean (false);
  file.seek (4);
System.out.println ("file.read boolean());
    file.close();
  }
  catch (Exception e)
  {
  System.out.println (e);
  }
}
```