Multithreading

Multitasking

Executing multiple tasks simultaneously.

Each task consists of cpu bound instructions and I/O bound instructions.

During I/O instruction execution, the cpu(processor) will be idle.

Multi tasking utilizes this time by assigning others tasks to the processor.

By making processor busy, the throughput (number of jobs that can be completed in unit time) can be increased.

OS will use different cpu scheduling algorithms to share processor time among multiple tasks.

A task can be atleast a program. We cant further subdivide.

Advantages

Reduces idle time of the cpu (increase throughput)

Quick response time.

Multithreading

Specialized form of multitasking.

Each task can be a thread.

A program can consists of two or more threads that can run concurrently.

Each ***thread*** defines a separate path of execution.

Threads are light weight, they share the same address space.

Context switching is easier.

Java provides built in support for multithreading.

In java the execution of main method itself is a thread created automatically when the program started.

Example_Program: To demonstrate main thread

```
class demo0
{
public static void main(String args[])
{
 Thread t=Thread.currentThread();
 System.out.println(t);
 t.setName("thread1");
 System.out.println(t);
 for(int i=1;i<=5;i++)
 {
   try
    {
      Thread.sleep(2000);
    }
   catch(InterruptedException e)
    {
      System.out.println(e);
    }
    System.out.println("i = "+i);
 }
}
}
```

Creating a thread

We can create a new thread in two ways

1. by extending Thread class.
2. by implementing Runnable interface.

With Runnable interface

Create a class by implementing Runnable interface.

Ex: class A implements Runnable

The Runnable interface has only one method

public void run()

Provide implementation for that method (ie write business logic)

Create the Thread instance as

```
Thread t= new Thread(new A());
```

the argument to the Thread constructor is an object of the class that implements Runnable interface.

To start the excution of your thread, call **t.start()**.

This invokes the **run()** method of the thread to execute the business logic of the thread.

Example_Program: Write a program to create a thread by implementing interface Runnable

```
class MyThread implements Runnable
{
   public void run()
   {
      for(int i=0;i<10;i++)
      {
         System.out.println("Child Thread:"+i);
      }
   }
}
class demo2
{
   public static void main(String args[])
   {
      MyThread m=new MyThread();
      Thread t1=new Thread(m);
      t1.start();
      for(int i=0;i<10;i++)
      {
            System.out.println("Main Thread:"+i);
        }
    }
}
```

Example_Program: Write a program to create a thread by implementing interface Runnable-adding sleep()

```
class MyThreaEd implements Runnable
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            try
            {
               Thread.sleep(500);
            }
            catch(InterruptedException e)
            {
               System.out.println(e);
            }
            System.out.println("Child Thread:"+i);
        }
    }
}
class demo3
{
    public static void main(String args[])
    {
        MyThread m=new MyThread();
        Thread t1=new Thread(m);
        t1.start();
        for(int i=0;i<10;i++)
        {
           try
           {
             Thread.sleep(500);
           }
           catch(InterruptedException e)
           {
             System.out.println(e);
           }
           System.out.println("Main Thread:"+i);
        }
    }
}
```

Extending Thred class

Create a subclass of Thread class.

Provide implementation for the run method.

To start the execution of the thread, call start() on the subclass object.

sleep()

Syntax: sleep(long millisec);

Static method

Makes the current thread to suspend its execution for specified millis secods.

If the thread is interrupted before the sleep time expires, it throws InterruptedException.

Example_Program : program to create a thread by extending Thread class.

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
          System.out.println("Child Thread :"+i);
        }
    }
}

class demo1
{
    public static void main(String args[])
    {
        MyThread m=new MyThread();
        m.start();
        for(int i=0;i<10;i++)
        {
            System.out.println("Main Thread:"+i);
        }
    }
}
```

Creating and Working with Multiple Threads

Example_Program: program to display 5th and 6th mathematical tables by creating two threads.

```
class FifthTable extends Thread
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("5 *"+i+" = "+(5*i));
        }
    }
}

class SixthTable extends Thread
{
    public void run()
    {
        for(int i=1;i<=10;i++)
        {
            System.out.println("6 *"+i+" = "+(6*i));
        }
    }
}

class demo4
{
    public static void main(String args[])
    {
        FifthTable f=new FifthTable();
        SixthTable s=new SixthTable();
        f.start();
        s.start();
    }
}
```

Thread priorities

An integer value that specify relative priority of one thread to another.

Among the threads of equal priority, JRE (Thread shedular) may schedule threads in any order for execution.

Methods:

    setPriority(int priority)

    int getPriority()

Priority value ranges from 0(low) to 10(high).

Normal priority is 5.

These are represeted by final static variables in Thread class

    Thread.MIN_PRIORITY

    Thread.MAX_PRIORITY

    Thread.NORM_PRIORITY

Thread schedular may give preference to high priority threads while scheduling threads for execution.

Thread priorities are only to influence the thread schedular.

Can't rely on them.

Example_Program: To demonstrate Thread Priorities.

```
class demo5 extends Thread
{
  public void run()
  {
    System.out.println("child thread priority"+Thread.currentThread().getPriority());

  for(int i=0;i<10;i++)
      System.out.print("\nchild thread "+i);
  }

  public static void main(String[] args)
  {
     demo5 t1 = new demo5();
```

```
        t1.setPriority(10);
        t1.start();

       System.out.println(Thread.currentThread().getPriority());
        for(int i=0;i<1000;i++)
           System.out.print("\nMain thread "+i);
    }
}
```

Context switching

A thread can voluntarily relinquish control.

A thread can be preempted by a higher priority thread.

yield()

Pauses the currently executing thread temporarily for giving a chance to the remaining threads of the same priority to execute.

If there is no waiting thread or all other waiting threads have a lower priority then the same thread will continue its execution.

The yielded thread when it will get the chance for execution is decided by the thread scheduler.

Example_Program : program to demonstrate yield() method.

```
class  MyThread extends Thread
{
   public void run()
   {
      for(int i=0;i<10;i++)
      {
         System.out.println("Child Thread "+i);
         if(i==5)
            Thread.yield();
      }
   }
}

class demo7
{
    public static void main(String args[])
    {
        MyThread m=new MyThread();
        m.start();
        for(int i=0;i<100;i++)
        {
            System.out.println("Main Thread "+i);
        }
    }
}
```

Other thread methods

isAlive() : helps us to know whether the thread has finished its execution or not.

join() : makes the the *caller thread* to wait until the *thread on which join()* has invoked completes it execution.

Example_Program : program to demonstrate join() method.

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=0;i<10;i++)
        {
            System.out.println("Child Thread");
            try
            {
                Thread.sleep(500);
            }
            catch(InterruptedException ie)
            {
              System.out.println(e);
            }
        }
    }
}
class demo6
{
    public static void main(String args[]) throws InterruptedException
    {
        MyThread t=new MyThread();
        t.start();
        t.join();
        for(int i=0;i<10;i++)
        {
            System.out.println("Main Thread");
        }
    }
}
```

Example_Program : creating a thread by making main clas to extend from Thread class.

```
class demo20 extends Thread
{
  public void run()
  {
   System.out.print("running");
  }
 public static void main(String[] args)
  {
   new demo20().start();
  }
}
```

Example_ Program : Creaing thread using inner classes.

```
class demo21
{
 public static void main(String[] args)
  {
     Runnable r=new Runnable()
             {
              public void run()
               { System.out.println("running");}
             };
     Thread t=new Thread(r);
     t.start();
  }
}
```

synchronization

Wihen two or more threads needs access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called synchronization.

Monitor

Key to synchronization is the concept of monitor.

A monitor is an object that is used as a mutually exclusive lock.

Only one thread can own an object's monitor at a given time.

When a thread acquires a lock, it is said to have entered the monitor.

All other threads attempting to enter the locked monitor will be suspended until the first thread exits the monitor.

All object have implicit monitor.

How to synchronize

We can synchronize a shared resource in two ways

1. synchronized methods.
2. synchronized block.
   (synchronized statement)

Method synchronization

Whichever the methods of the resource(object) you want to synchronize, decalare those methods with synchronized modifier.

All objects have implicit monitor.

To enter an objects monitor, just call any synchronized method.

Note:

While a thread is inside a synchronized method, all other threads that try to call it (or any other synchronized method) on the same object have to wait.

Example_Program : Program where two threads processes the same resource (unsynchronized process).

```
class Display
{
    public  void wish(String name)
    {
         for(int i=0;i<10;i++)
         {
             System.out.print("Good morning");
             try
             {
                 Thread.sleep(2000);
             }
             catch(InterruptedException e)
             {
             }

              System.out.println(name);

          }
       }
}
class Mythread extends Thread
{
     Display d;
     String name;
```

```
      Mythread(Display d,String name)
      {
            this.d=d;
            this.name=name;
      }

      public void run()
      {
          d.wish(name);
      }
}
class demo9
{
           public static void main(String args[])
           {
                 Display d=new Display();
                 Mythread t1= new Mythread(d,"RAM");
                 Mythread t2= new Mythread(d,"LAXMAN");
                 t1.start();
                 t2.start();
           }
}
```

Example_Program : To demonstrate synchronized methods.

```
class Display
{
     public synchronized void wish(String name)
     {
           for(int i=0;i<10;i++)
           {
                System.out.print("Good morning : ");
                try
                {
                     Thread.sleep(2000);
                }
                catch(InterruptedException e)
                {
                }

                 System.out.print(name);
                 System.out.println();

            }
```

```
        }
}
class Mythread extends Thread
{
       Display d;
       String name;
       Mythread(Display d,String name)
      {
              this.d=d;
              this.name=name;
       }

       public void run()
       {
            d.wish(name);
       }
}
class demo10
{
             public static void main(String args[])
             {
                    Display d=new Display();
                    Mythread t1= new Mythread(d,"RAM");
                    Mythread t2= new Mythread(d,"LAXMAN");
                    t1.start();
                    t2.start();
             }
}
```

Synchronized block

If you want to synchronize access to objects of a class that was not designed for multithreaded access (that is the class does not use snchronzed methods).

If the class was created by a third party, we do not have access to the code.

Then we can acquire lock on the object with synchronized block.

syntax

```
synchronized(target_instance)

   {

     target_instance.method1();

   }
```

<u>Example_Program :</u> Synchronized block example.

```
class Display
{
    public  void wish(String name)
    {
      synchronized(this)
      {
        for(int i=0;i<10;i++)
        {
            System.out.print("Good morning : ");
            try
            {
                Thread.sleep(2000);
            }
            catch(InterruptedException e)
            {
            }
           System.out.print(name);
           System.out.println();
         }
      }
    }
}

class Mythread extends Thread
{
     Display d;
     String name;
     Mythread(Display d,String name)
     {
          this.d=d;
        this.name=name;
     }

     public void run()
     {
        d.wish(name);
     }
}
class demo11
{
        public static void main(String args[])
        {
             Display d=new Display();
             Mythread t1= new Mythread(d,"RAM");
             Mythread t2= new Mythread(d,"LAXMAN");
             t1.start();
             t2.start();
        }
}
```

Class level locking

Class level locking prevents multiple threads to enter in synchronized block in any of all available **instances** on runtime.

This means if in runtime there are 100 instances of a Class, then only one thread will be able to execute that code in any one of instance at a time, and all other instances will be locked for other threads.

This should always be done to make static data thread safe.

Class level locking-Example 1

```
public class DemoClass
  {
     public void demoMethod()
```

```
      {
          synchronized (DemoClass.class)
            {
              //other thread safe code
            }
       }
  }
```

static synchronized methods-Example 2

```
public class DemoClass
  {
      public synchronized static void    demoMethod(){}
  }
```

Static synchronized methods will lock the class instead of object.

Note:
Static and non static synchronized methods will not block each other (ie they can run at the same time from different threads since they acquire lock on different things).

Inter Thread communication

public final void wait() throws InterruptedException

public final void wait(long timeout) throws InterruptedException

Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

public final void notify()

Wakes up a single thread that is waiting on this object's monitor.

If many threads are waiting on this object, one of them is chosen to be awakened.

The choice is arbitrary and occurs at the discretion of the implementation

public final void notifyAll()

wakes up all the threads that are waiting on this object's monitor.

One of the thread will be granted access by the Thread Schedular.

These three methods are not from Thread class, these are from Object class.

Note:

1) There exists a very rare possibility that the waiting thread resumes without notify(). (for no apparent reason)

Oracle recommends that calls to wait() should take place with in a loop that checks the condition on which the thread is waiting.

2) IllegalMonitorStateException

Thrown to indicate that a thread has attempted to **wait** on an object's monitor Or to **notify** other threads waiting on an object's monitor without owning the specified monitor.

3) You must call the wait(), notify() or notifyAll() from a synchronized context.

Example_Program : This program is used to show the inter thread communication.

```
 class Buffer
{
int a;
boolean produced = false;

public synchronized void produce(int x)
{

if(produced)
 {
   System.out.println("Producer is waiting...");
   try{
     wait();
   }catch(Exception e){
   System.out.println(e);
 }
}
a=x;
System.out.println("Product" + a + " is produced.");
produced = true;
notify();
}

public synchronized void consume()
{
if(!produced)
 {
  System.out.println("Consumer is waiting...");
  try{
   wait();
     }catch(Exception e){
   System.out.println(e);
         }
 }
System.out.println("Product" + a + " is consumed.");
produced = false;
notify();
}
}

class Producer extends Thread
{
Buffer b;
public Producer(Buffer b)
{
 this.b = b;
}
public void run()
{
 System.out.println("Producer start producing...");
 for(int i = 1; i <= 10; i++)
 {
   b.produce(i);
 }
}
}


class Consumer extends Thread{
Buffer b;
public Consumer(Buffer b){
this.b = b;
}
```

```
public void run()
{
  System.out.println("Consumer start synchronized    consuming...");
  for(int i = 1; i <= 10; i++)
      {
   b.consume();
      }
}
}


public class demo12
{
public static void main(String args[])
{
//Create Buffer object.
Buffer b = new Buffer();
//creating producer thread.
Producer p = new Producer(b);
//creating consumer thread.
Consumer c = new Consumer(b);
//starting threads.
p.start();
c.start();
}
```