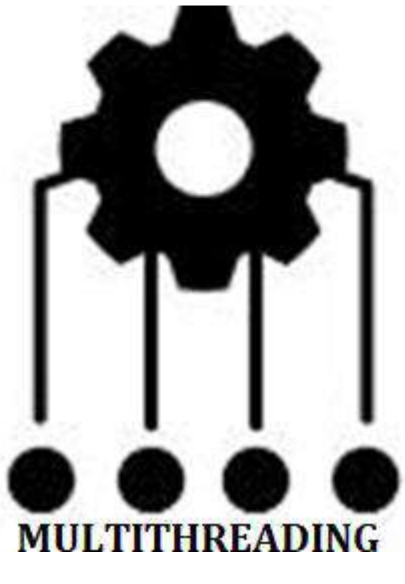
# CO – 3



## SHORT ANSWER QUESTIONS ( from CO-3 )

**1. What is parallel programming?**

*Answer:*

Parallel programming can also be called as Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

- Process-based Multitasking (Multiprocessing)
- Thread-based Multitasking (Multithreading)

**2. Identify the differences between Multithreading and Multitasking**

*Answer:*

| | Process-based Multitasking | Thread-based Multitasking |
|---|---|---|
| Address space | Each process have its own address in memory i.e. each process allocates separate memory area. | Threads share the same address space. |
| weight | Process is heavyweight. | Thread is lightweight. |
| Cost of communication | Cost of communication between the process is high. | Cost of communication between the thread is low. |

## 3. Threads and Thread states?

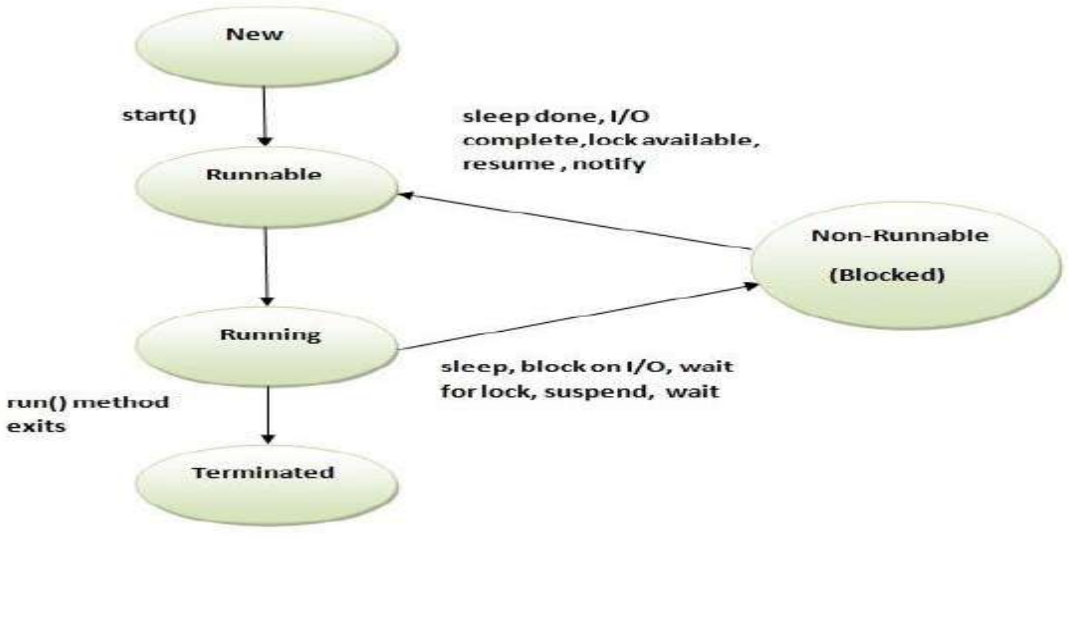
*Answer:*

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area.

The life cycle of the thread with thread states are as follows:

1. New : The thread is in new state if you create an instance of Thread class but before the invocation of start() method.
2. Runnable : The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3. Running: The thread is in running state if the thread scheduler has selected it.
4. Non-Runnable (Blocked): This is the state when the thread is still alive, but is currently not eligible to run.
5. Terminated: A thread is in terminated or dead state when its run() method exits.


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

**4. List the ways, threads can be declared in Java?**

*Answer:*

Threads in Java can be declared in two ways.

1. Extends Thread

```
Class Thr1 extends Thread{
Public void run(){
   }
}
```

2. Implements Runnable

```
  Class Thr2 implements Runnable{
     Public void run(){
        }
     }
```

**5.Illustrate Thread class with fully written constructor.**

*Answer:*

```
   Class MyThread extends Thread {
   String str;
   MyThread(String s){
   Str=s;
    }
}
```

**6.Discuss the importance of run() method?**
*Answer:*

**public void run()** is used to perform action for a thread.

Run() is the main method of java thread program, thread behavior declared in this method. After creating the thread object,

```
Thread o1 = new Thread();
```

`o1.start();` Would call the run() method.

**7.Explain sleep() method in threads.**
*Answer:*

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

Eg:

public static void sleep(long miliseconds) throws InterruptedException

**8. What is the purpose of join()?**
*Answer:*

The join() method waits for a thread to die. In other words, it causes the currently running threads to stop executing until the thread it joins with completes its task.

**9.Judge the relevance of priority in threads?**

*Answer:*

Each thread have a priority. Priorities are represented by a number between 1 and 10. In most cases, thread schedular schedules the threads according to their priority (known as preemptive scheduling). But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

Eg:

```
Thread t1 = new Thread();

t1.setPriority(6);

Int g = t1.getPriority();
```

the above two methods available.

**10.Types of Synchronization in java.**

*Answer:*

It is to maintain the critical section among threads.

Key word: synchronized

1. Synchronized method()
   Eg. synchronized void add(){

   }

2. Synchronized block   - from any method

   Eg. synchronized(this){
                                  }

**11. Differentiate synchronized block and method**

*Answer:*

| | **Synchronized method** | **Synchronized block** |
|---|---|---|
| Lock | Object/class level | Only object level |
| scope | Entire method | Limited to few statements |
| Performance | low | high |
| Waiting time | high | low |
| Null pointer exception | no | yes |

## LONG ANSWER QUESTIONS ( from CO-3 )

**1. Java Thread program for displaying “HelloWorld”.**

*Answer:*

```
Classs MyThr implements Runnable {
public void run(){
System.out.println(“Hello World”);
   }
public static void main(String[] args){
MyThr m1 = new MyThr();
Thread t1 = newThread(m1);
t1.start();
    }
}
```

**2. Java thread to print 1 to 10 numbers…**

*Answer:*

```
Classs MyThr implements Runnable {
public void run(){
for(int I = 1; i<=10; i++)
System.out.println(i);
   }
public static void main(String[] args){
MyThr m1 = new MyThr();
Thread t1 = newThread(m1);
t1.start();
    }   }
```

**3. Sleep() method -extended from Thread .**
*Answer:*

```
classs MyThr extends Thread {
public void run(){
try{
for(int I = 1; i<=10; i++)
Thread.sleep();
System.out.println(i);
} catch(Exception e){
System.out.println(e);
}
   }
public static void main(String[] args){
MyThr t1 = new MyThr();
MyThr t2 = new MyThr();

t1.start();
t2.start();
    }
}
```

4. **Java program- thread1 is derived from Thread class which should produce numbers from 1 to 10, thread2 is drawn from Runnable interface to produce numbers from 10 to 1. Demo class to output both threads created.**

*Answer:*

```
classs MyThr1 extends Thread {
public void run(){
try{
for(int I = 1; i<=10; i++)
Thread.sleep();
System.out.println(i);
} catch(Exception e){
System.out.println(e);
    }
  }
}
Class MyThr2 implements Runnable {
public void run(){
try{
for(int i = 10; i>=1; i--)
Thread.sleep();
System.out.println(i);
} catch(Exception e){
System.out.println(e);
}
   }
}
```

```
Class Demo {
public static void main(String[] args){
MyThr1 t1 = new MyThr1();
MyThr2 m2 = new MyThr2();
Thread t2 = new Thread(m2);
t1.start();
t2.start();
    }
}
```

5. **Java thread  program –Runnable interface - to print Fibonacci sequence numbers / even odd numbers**

   Note: This program is good for any three/five thread methods of thread ( sleep(), start(), isAlive(), join() and run() methods )

*Answer:*

```
class FiboSeq{
void fibo(){
int a,b,f;
a=0;
b=1;
System.out.println(a);
System.out.println(b);
try{
for(int i=0; i< 8; i++){
f=a+b;
System.out.println(f);
Thread.sleep(500);
a=b;
b=f;
   }
}
catch(Exception e){
System.out.println(e);
}
 }

void evenOdd(){
int[] ar = {10,23,45,20,24,99};
for(int i=0; i<ar.length; i++){
if ((ar[i] % 2) == 0)
System.out.println(ar[i]+ "  "+ " is Even number");
else
System.out.println(ar[i]+ "  "+ " is ODD number");
   }
 }
}

class ThrN implements Runnable{
FiboSeq fs;
ThrN(FiboSeq f1){
fs=f1;
}
```

```
public void run(){
fs.fibo();
}
}

class ThrM implements Runnable{
FiboSeq fs;
ThrM(FiboSeq f1){
fs=f1;
}

public void run(){
fs.evenOdd();
}
}
class ThrFibo1 {
public static void main(String[] args){
FiboSeq ff = new FiboSeq();

ThrN tt = new ThrN(ff);
Thread t1 = new Thread(tt);

ThrM tk = new ThrM(ff);
Thread t2 = new Thread(tk);

t1.start();
t2.start();
System.out.println(" t1 is existing? "+ t1.isAlive());
System.out.println(" t2 is existing? "+ t1.isAlive());
try{
t1.join();
t2.join();
}catch(Exception e){
System.out.println(e);
}

System.out.println(" t1 is existing? "+ t1.isAlive());
System.out.println(" t2 is existing? "+ t1.isAlive());
 }
}
```

6. **Develop a java program, your program should have two threads with two different purposes to demonstrate start(), run(), sleep(), getPriority(), setPriority() methods.**

( Count of even/odd numbers and count of zeroes/Non-zeroes are possible in this program )

*Answer:*

```
class Thr1 extends Thread {
String str;
Thr1( String str){
this.str =str;
}

public void run(){
int ectr=0;
int octr=0;


int ar[] = {12,97,87,54,78,32,9,91,53,75};
for(int i=0;  i<ar.length;  i++){
if((ar[i] % 2) == 0) {
System.out.println(str +":   "+ ar[i] + "   is  an  EVEN
number");
ectr++;
}
else {
System.out.println(str  +  "  :  "+ar[i]  +  "   is  an  ODD
number");
octr++;
}
}
System.out.println(str  +":   count  of  EVEN  numbers  =
"+ectr);
System.out.println(str  +":   count  of  ODD  numbers  =
"+octr);
}
}


class Thr2 implements Runnable {
String str;
Thr2( String str){
this.str =str;
}
```

```
public void run(){
int zctr=0;
int nzctr=0;
try{
int ar[] = {12,0,87,54,0,32,9,91,53,0};
for(int i=0;  i<ar.length;  i++){
if(ar[i]  == 0) {
System.out.println(str +":  "+ ar[i] + "  is a ZERO");
zctr++;
}
else {
System.out.println(str + ":  "+ar[i] + "  is NON ZERO");
nzctr++;
}
Thread.sleep(500);
}
} catch(Exception e){
System.out.println(e);
}
System.out.println(str  +"    count  of  ZERO  numbers  =
"+zctr);
System.out.println(str  +"   count  of  NonZero  numbers  =
"+nzctr);
}
}
```

```
class ThrDemo5 {
public static void main(String[] args){
String s1 = "FIRST";
String s2 = "SECOND";

Thr1 t1 = new Thr1(s1);
Thr1 t3 = new Thr1(s1);

Thr2 gg = new Thr2(s2);
Thread t2 = new Thread(gg,s2);

t1.start();
t2.start();
t3.start();

System.out.println(" Name of Thread1 ="+ t1.getName());
System.out.println(" Name of Thread2 ="+ t2.getName());

t1.setPriority(9);
t3.setPriority(2);
System.out.println(t1.getName()  +  "  priority  :  "  +
t1.getPriority());
System.out.println(t3.getName()  +  "  priority  :  "  +
t3.getPriority());

System.out.println(" BEFORE calling join() method");
System.out.println(t1.getName()  +  "  is  exists?  "  +
t1.isAlive());
System.out.println(t2.getName()  +  "  is  exists?  "  +
t2.isAlive());
System.out.println(t3.getName()  +  "  is  exists?  "  +
t3.isAlive());


try{
t1.join();
t2.join();


t3.java();
}catch(Exception er){
System.out.println(er);
}
System.out.println(" AFTER calling join() method");
System.out.println(t1.getName()  +  "  is  exists?  "  +
t1.isAlive());
System.out.println(t2.getName()  +  "  is  exists?  "  +
t2.isAlive());
System.out.println(t3.getName()  +  "  is  exists?  "  +
t3.isAlive());


 }
}
```

7. **Describe synchronization in java threads, write a java program that included a synchronized block in a particular method for a genuine reason.**

*Answer:*

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

*Program:*

```
class Bangee{
 void doit(String str){
// synchronized block
synchronized(this){
try{
System.out.print("[ "+str);
Thread.sleep(1000);
}
catch(Exception e){
System.out.println(e);
}

System.out.println("]");
```

```
        }
    }
   }

   class Thr1 implements Runnable {
   Bangee bg;
   String str;
   Thr1(Bangee bb, String str1){
   bg=bb;
   str = str1;
   }
   public void run(){
   bg.doit(str);
   }
   }

   class TsyncDemo1{
   public static void main(String[] args){
   Bangee bg1 = new Bangee();
   Thr1 tt1 = new Thr1(bg1,"raja");
   Thr1 tt2 = new Thr1(bg1,"Kaja");
   Thread t1 = new Thread(tt1);
   Thread t2 = new Thread(tt2);
   t1.start();
   t2.start();
   }
   }
```

**8. A java program to implement synchronized method**

*Answer:*

```
class Bangee{
// synchronized method…
synchronized void doit(String str){
try{
System.out.print("[ "+str);
Thread.sleep(1000);
}
catch(Exception e){
System.out.println(e);
}

System.out.println("]");
}
}

class Thr1 implements Runnable {
Bangee bg;
String str;
Thr1(Bangee bb, String str1){
bg=bb;
str = str1;
}
public void run(){
bg.doit(str);
}   }
```

```
class TsyncDemo1{
public static void main(String[] args){
Bangee bg1 = new Bangee();
Thr1 tt1 = new Thr1(bg1,"raja");
Thr1 tt2 = new Thr1(bg1,"Kaja");
Thread t1 = new Thread(tt1);
Thread t2 = new Thread(tt2);
t1.start();
t2.start();
}
}
```

**9. Lock Interface and Rentrant Lock**

Program 1:

```
// Lock and ReentrantLock  - synchronization program...
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

class TableX{
Lock qLock = new ReentrantLock();
void printX(int n){
 qLock.lock();
try{
for (int i= 1; i<=10; i++){
System.out.println( n +" X "+ i +" = "+n*i);
Thread.sleep(1000);
}
}catch(Exception e){
System.out.println(e);
  }
      qLock.unlock();
       }
}

class ThrTable extends Thread {
TableX tbl;
int n;
ThrTable(TableX tbl1, int n1){
tbl=tbl1;
n=n1;
}

public void run(){
tbl.printX(n);
}
}

class Thr12Lock1{
public static void main(String[] args){
TableX tx = new TableX();
ThrTable t1 = new ThrTable(tx, 5);
ThrTable t2 = new ThrTable(tx, 10);
t1.start();
```

```
t2.start();
 }
}
```

Program 2:

```
// Lock and ReentrantLock  - synchronization program...

import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
class StrX{
Lock qLock = new ReentrantLock();
void printX(String str){
qLock.lock();
try{
System.out.print("[ "+ str);
Thread.sleep(500);
System.out.println(" ]");
}catch(Exception e){
System.out.println(e);
}finally{
   qLock.unlock();
  }
}
}
```

```
class ThrStr extends Thread{
StrX sx;
String s;
ThrStr( StrX sx1, String s1){
sx = sx1;
s = s1;
}

public void run(){
sx.printX(s);
}
}

class Thr13Lock2{
public static void main(String[] args){

StrX sx = new StrX();
ThrStr t1 = new ThrStr(sx, "KLEF is");
ThrStr t2 = new ThrStr(sx, "world class");
ThrStr t3 = new ThrStr(sx, "University");
t1.start();
t2.start();
t3.start();
  }
}
```

**10. Semaphore – programs**

Program 1:

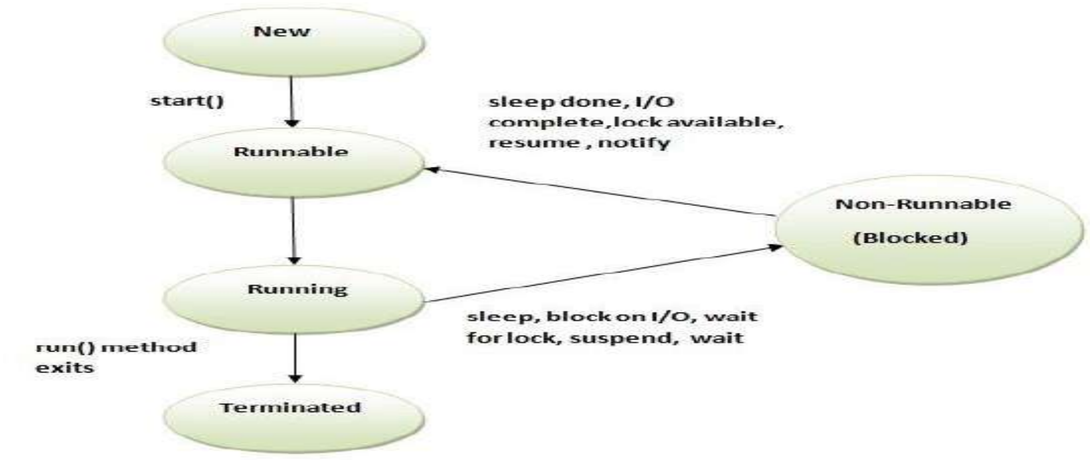
```
import java.util.concurrent.*;
 // Thread coodination - with semaphore
class TableX{
Semaphore sem = new Semaphore(1);

void printX(int n){
try{
sem.acquire();
System.out.println( "  ");
for (int i= 1; i<=10; i++){
System.out.println( n +" X "+ i +" = "+n*i);
Thread.sleep(1000);
}
}catch(Exception e){
System.out.println(e);
  }
      sem.release();
   }
}

class ThrTable extends Thread {
     TableX tbl;
     int n;
    ThrTable(TableX tbl1 , int n1)
    {
        tbl = tbl1;
        n=n1;
    }

public void run(){
tbl.printX(n);
}
}

class Thr14Semaphore1{
public static void main(String[] args){
TableX tx = new TableX();
ThrTable t1 = new ThrTable(tx, 5);
ThrTable t2 = new ThrTable(tx, 10);
t1.start();
t2.start();
 }
}
```

Program 2:

```
import java.util.concurrent.*;
 // Thread coodination - with semaphore
class StrX{
Semaphore sem = new Semaphore(1);
void printX(String str){

try{
sem.acquire();
System.out.print("[ "+ str);
Thread.sleep(500);
System.out.println(" ]");
}catch(Exception e){
System.out.println(e);
}
   sem.release();

   }
}

class ThrStr extends Thread{
StrX sx;
String s;
ThrStr( StrX sx1, String s1){
sx = sx1;
s = s1;
}

public void run(){
sx.printX(s);
}
}

class Thr15Semaphore2{
public static void main(String[] args){

StrX sx = new StrX();
ThrStr t1 = new ThrStr(sx, "KLEF is");
ThrStr t2 = new ThrStr(sx, "world class");
ThrStr t3 = new ThrStr(sx, "University");
t1.start();
t2.start();
t3.start();
}
}
```

**11. Examine the process of inter-thread communication (communication between threads) with a neat diagram, explain wait() and notify() methods in this context.**

*Answer:*


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

Wait() - The *wait()* method causes the current thread to wait indefinitely until another thread either invokes *notify()* for this object .

Notify() - For all threads waiting on this object's monitor (by using any one of the *wait()* methods), the method *notify()* notifies any one of them to wake up arbitrarily.

Program:

```
// wait() and notify() demo program...


class Data {
        boolean sig = false;
        synchronized void part1() {
        System.out.println("Welcome to KLEF");
        sig = true;
        notify();
    }
    synchronized void part2() {
        try {
            if (!sig)
               wait();
                }   catch (Exception e) {
                   System.out.println(e);
              }
                System.out.println("Join in any one Tech
club");
    }
}

class Thr1 extends Thread {
Data dt;
Thr1(Data dt1){
dt = dt1;
}

public void run() {
dt.part1();
}
}

class Thr2 extends Thread {
Data dt;
Thr2(Data dt1) {
```

```
dt = dt1;
}

public void run() {
dt.part2();
}
}

 class Thr16WtNtf1{
     public static void main(String[] args) {
         Data dt = new Data();

        Thr1  t1 = new Thr1(dt);
        Thr2  t2 = new Thr2(dt);

        // Start t2 and then t1
       t2.start();
       t1.start();

    }
}
```

**Program 2:**

```
class Bank{
boolean sig = false;

synchronized void deposit(){
```

```
System.out.println(" your deposit is successful");
System.out.println(" you can withdraw any time");
System.out.println("  ");
sig = true;
notify();
}

synchronized void withdraw(){
try{
if (!sig)
wait();
} catch(Exception e){
System.out.println(e);
}
System.out.println(" Thank you for withdrawing your
money");
 }
}

class Thr1 extends Thread{
Bank bk;
Thr1(Bank bk1){
bk=bk1;
}
public void run(){
bk.deposit();
}
```

```
}

class Thr2 extends Thread{
Bank bk;
Thr2(Bank bk1){
bk=bk1;
}
public void run(){
bk.withdraw();
}
}

class Thr17WtNtf2{
public static void main(String[] args){
Bank bkk = new Bank();
Thr1 t1 = new Thr1(bkk);
Thr2 t2 = new Thr2(bkk);

t2.start();
t1.start();
 }
}
```

## 12. Deadlock program (situation how deadlock occurs)

**Program**:

```
class BookTkt extends Thread {
Object train, comp;
BookTkt(Object t1, Object c1){
train = t1;
comp=c1;
}

public void run(){
synchronized(train){
System.out.println(" Book ticket locked on train");
try{
Thread.sleep(150);
}catch(Exception e){
System.out.println(e);
}
System.out.println(" Book ticket now waiting to lock on
compartment");
    synchronized(comp){
   System.out.println(" Book ticket locked on
compartment");
  }
}
}
}
```

```
class CancelTkt extends Thread{
Object train, comp;
CancelTkt(Object t1, Object c1){
train = t1;
comp=c1;
}
public void run(){
synchronized(comp){
System.out.println(" cancel ticket locked on 
compartment");
try{
Thread.sleep(200);
}catch(Exception e){
System.out.println(e);
}
System.out.println(" cancel ticket now waiting to lock 
on train");
    synchronized(train){
   System.out.println(" cancel ticket locked on train");
  }
}
}
}

class Thr19DeadLock{
public static void main(String[] args){
Object train = new Object();
Object compartment = new Object();
BookTkt obj1 = new BookTkt(train, compartment);
CancelTkt obj2 = new CancelTkt(train, compartment);

Thread t1 = new Thread(obj1);
Thread t2 = new Thread(obj2);
t1.start();
t2.start();
 }
}
```

**13. Avoiding Deadlock ( program to handle deadlock)**

**Program:**

```
class BookTkt extends Thread {
Object train, comp;
BookTkt(Object t1, Object c1){
train = t1;
comp=c1;
}

public void run(){
synchronized(train){
System.out.println(" Book ticket locked on train");
try{
Thread.sleep(150);
}catch(Exception e){
System.out.println(e);
```

```
}
System.out.println(" Book ticket now waiting to lock on
compartment");
    synchronized(comp){
   System.out.println(" Book ticket locked on
compartment");
  }
}
}
}

class CancelTkt extends Thread{
Object train, comp;
CancelTkt(Object t1, Object c1){
train = t1;
comp=c1;
}
public void run(){
synchronized(train){
System.out.println(" cancel ticket locked on train");
try{
Thread.sleep(150);
}catch(Exception e){
System.out.println(e);
}
System.out.println(" cancel ticket now waiting to lock
on compartment");
    synchronized(comp){
```

```
   System.out.println(" cancel ticket locked on
compartment");
  }
}
}
}

class Thr20DeadLockAvoid{
public static void main(String[] args){
Object train = new Object();
Object compartment = new Object();
BookTkt obj1 = new BookTkt(train, compartment);
CancelTkt obj2 = new CancelTkt(train, compartment);

Thread t1 = new Thread(obj1);
Thread t2 = new Thread(obj2);
t1.start();
t2.start();
 }
}
```

**14. A classical Producer and Consumer java threads to demonstrate inter thread communication.**

*Program :*

```
class Data {
int n;
boolean sig=false;

synchronized void put(int n1) {
  try {
        if(sig)
        wait();
 }catch(Exception e) {
   System.out.println(e);
  }

n=n1;
System.out.println("put= "+n);
sig=true;
notify();
  }

synchronized void get() {
   try {
       if(!sig)
        wait();
 }catch(Exception e) {
   System.out.println(e);
```

```
    }
  System.out.println("Got= "+n);
    sig=false;
    notify();
    }
  }

  class Producer extends Thread {
  Data dt;
  Producer(Data dt1){
  dt=dt1;
  }

  public void run() {
  int i=1;
  for(int lp=0; lp<20; lp++)
  dt.put(i++);
     }
  }

  class Consumer extends Thread {
  Data dt;
  Consumer(Data dt1) {
  dt=dt1;
  }

  public void run() {
```

```
 for(int lp=0; lp<20; lp++)
 dt.get();
  }
}

class Thr18ProdConsumer {
public static void main(String args[]) {
Data dt= new Data();
Producer p1 = new Producer(dt);
Consumer c1 = new Consumer(dt);
p1.start();
c1.start();
 }
}
```