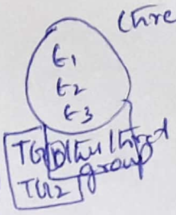
ThreadGroup → java.lang.ThreadGroup

Constructors → ThreadGroup(String name)
creats tgroup with givan name

ThreadGroup(ThreadGroup parent,
String name);
Creats tgroup with parent tgroup and name



} suspend
resume
Interrupt
group of threads
by single method call

Deprecated method

boo allowThreadSuspension(boo b)
resume()
stop(.), suspend()

How to create Thread Group

Create Parent Thread

↓

Adding Threads to Thread Group

↓

Create child ThreadGroup

↓

Add the Threads into child ThreadGroup

ThreadGroup pG = new ThreadGrp("Pg");
Thread t1 = new Thread(Pg, "t1");
Thread t2 = new Thread(Pg, "t2");

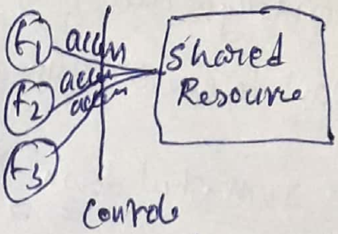
ThreadGroup cG = new ThreadGroup(PG, "child");
Thread t3 = new Thread(cG, "t3");

int activeCount()
int activeGroupCount()
int getMaxPriority()
void setMaxPriority(int pri)
String toString()
boolean isDaemon()

void setDemon(boolean b)
void list()
String getName()
ThreadGroup getParent()
void interrupt()
int enumerate(ThreadGroup[] TG)

boolean isDestroyed()
boolean parentOf(ThreadGroup p)
void checkAccess()
void destroy()
int enumrate(Thread[] list)
int enumerate(Thread[] list, boolean recurse);

Syncronization:- the capability to controle the access of multiple threads to any shared resources.



Uses :- to prevent thread interface
to prevent consistency problems

Thread Interface in java:- is a condition which occurs when more than one threads, executing simultaneously, access same piece of data. due to this possible that data may get corrupted @ one may not get the desired output.

When it occurs -> code written in not thread safe way

How to avoid thread Interface
@
How to achive Thread Safeness
- -> By declaring the method as Syncronized
- -> By declaring the variable as final
- -> By declaring the variable as Volatile
- -> By Creating the immutable objects
- -> By using Atomic operations.
- -> restricting the access to same object by multiple threads.

Not in thread Safe Code.

```
class A {
  int i;
  void sharedMeth(){
    i=10 s.o.p(i);
    i=20 s.o.p(i);
    i=30 s.o.p(i);}
  }
}
class Test {
  p s v m(String[] args){
  final A a=new A()
  Thread t1=new Thread()
  Thread t2=new Thread()
  t1.start()
  t2.start()
```

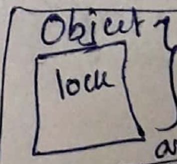
t1 t2 simultaneously accessing the run method

```
void run(){
  a.sharedMethod()
}
```

Types of Syncronization ① Process Syncronization ② Thread Syncronization

② **Thread Syncronization** ① Mutual Exclusive ② Inter-thread Communication

Mutual Exclusive -> Syncronized method
-> Syncronized block
-> Static Syncronization



every Object has a lock associated with it

Concept of Lock in java -> Syncronization is built around an internal entity known as the lock @ monitor. Every object has an lock with it
-> Thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them and then release the lock when it's done with them.

Lock (Interface) -> java.util.concurrent.locks

ReentrantLock -> Lock lock = new ReentrantLock();

ReentrantLock -> public class ReentrantLock extends Object implements Lock, Serializable.

Codesnippet

```
class X {    ~~Reent~~
private final Lock lock = new ReentrantLock();
    public void m() {
        lock.lock();
        try {
            // ... method body
        } finally {
            lock.unlock();
        }
    }}
```

Methods

```
void        lock()
void        unlock()
void        lockInterruptibly()
Condition   newCondition()
String      toString()
boolean     tryLock()
boolean     tryLock(long timeout, TimeUnit unit)
boolean     isFair();
boolean     isHeldByCurrentThread()
```

```
int getHoldCount()
protected Thread getOwner()
protected Collection<Thread>
        getQueuedThreads()
int getQueueLength()
protected Collection<
    <Thread> getWaitingThreads
        (Condition cond),
int getWaitQueueLength(Condition condition);
boolean hasQueuedThread(Thread thread)
boolean hasQueuedThreads();
boolean hasWaiters(Condition condition)
boolean
```

Mutual Exclusive ① Syncronized Method
② Syncronized~~...~~ block
③ Static syncronization.

Syncronized Method -> If you declare any method as Syncronized.

```
Ex  class A {
    Syncronized void printTable(int n)
      for(int i=1; i<5; i++)
        s.op(n*i);
      try { Thread.sleep(400);
      } catch(Exp){}
```

```
Ex  class A {
      syncronized void m1(int a) { // Syncronized Method
         - - - - -
         : .
      }}
```

```
class B extends Thread {
  A a;
  Thread t1 = new Thread(); t1.start
  Thread t2 = new Thread(); t2.start
  Th
```

Syncronized block in java :- Used to perform syncronization on any specific resource of the method. Ex 50 lines in code need to syncronize 5 lines need to use syncronization block. **Note:-** -> Syncronized block is used to lock an object for any shared Resource
-> scope of Syncronized block is smaller than the method

Syntax:-

```
syncronized (Object reference expression){
    // code block
}
```

```
Ex  class A {
    void print(int i){
        syncronize(this){
            for(int i=1; i<=20; i++){
                s.o.p(n*i);
        }
    }
```

Static Syncronization

-> If you make any Static method as Syncronized, the lock will be on the class not an object.

Syntax:

```
Syncronized static void method (int i){
    // code
}
```

Ex: Table{

```
Syncronized static void print(int i){
   for(int i=1; i<=10; i++){
        Syst.(n*i)
   }
Thread.print(10);
```

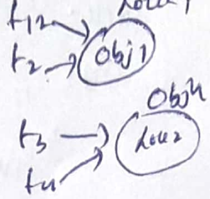
problem without Static Syncronization

two shared class named obj1, Obj2

Synchronized meth block, Count (Table)

Interface b/w t1 and t2 because refer common object.

Interface b/w t3 and t4 that have a single lock. If interface b/w



t1	t2
t3	t4

Loc Lock

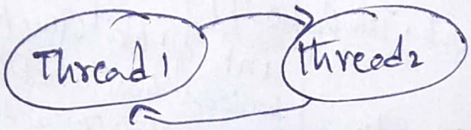
t1 t3 accaires differ lock.

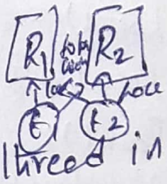
t2 t4 This solves by using

Loc Loc

Static Syncronization

Dead lock in java





Dead lock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread Second thread is waiting for an object lock that is acquired by first thread both threads are waiting for each other to release the lock, the condition is called Dead lock.

Ex

```
TestDead lock Example{
PSVM (String[] args){
final String resou1="ratan";
final Str reson2= "vimal";
Thread t1 = new Thread(){
   void run(){
      Syncronized (reson1){
            S.O.P("t1: lock R1");
      try { Thread.Sleep(100);} catch (Ex e);
         syncronized (reson2){
            S.O.P("t1: lock R t2") }}
```

```
Thread t2 = new Thread(){
   void run(){
      Syncronized (R2){
         S.O.P (t2: lock R2);
      try{ Thread.Sle(100);
      Syncronized(R1){
      S.O.P("t2: loc R1 t1");
      }
   }
```