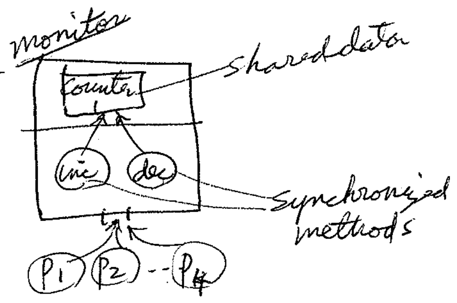
Park

```
Script started
zodiac:~/JAVA/Num > cat Numbers.java

//This is a test multithreaded program in which 5 threads are created
//and executed concurrently.
//Each thread gets its id number and displays it 100 times.

public class Numbers //in single file version, only main class is public
{ public static void main(String args[])
  {NumberThread num1, num2, num3, num4, num5; //5 threads

   num1 = new NumberThread(1); num1.start(); //creates & starts a thread
   num2 = new NumberThread(2); num2.start(); //creates & starts a thread
   num3 = new NumberThread(3); num3.start(); //creates & starts a thread
   num4 = new NumberThread(4); num4.start(); //creates & starts a thread
   num5 = new NumberThread(5); num5.start(); //creates & starts a thread
  }//main
}//Numbers

class NumberThread extends Thread
{ int num;
  public NumberThread(int n) {num = n;} //constructor
  public void run()
  { for (int k=0; k<100; k++)
     {System.out.print(num);
      }
   }//run
 }//NumberThread


zodiac:~/JAVA/Num > ls
Numbers.java  typescript
zodiac:~/JAVA/Num > javac Numbers.java
zodiac:~/JAVA/Num > ls
Numbers.class     Numbers.java    NumberThread.class    typescript
zodiac:~/JAVA/Num > java Numbers

1111111111111111111111111111111111111111111111111111111111111111111111
1111111111111111111111111111112222222222222222222222222222222222222222
2222222222222222222222222222222222222222222222222222222222223333333333
3333333333333333333333333333333333333333333333333333333333333333333333
3333333333333333333344444444444444444444444444444444444444444444444444
4444444444444444444444444444444444444444444444444455555555555555555555
5555555555555555555555555555555555555555555555555555555555555555555555
5555555555

zodiac:~/JAVA/Num > exit
exit
script done
```

```
//This program is a sample program for testing multithreading
//and shared-var access using monitor mechanism.
//Four threads are concurrently running with accessing a
//shared-object (s1).
//Shared-object has a private "counter" (shared-var) that is shared
//among multiple threads.  So, the shared-object is a monitor
//for providing mutex for the shared-var "counter".

public class sample //main class
//in the single-file version, only main class is public
{ static some s1 = new some(); //create one shared obj

  public static void main(String args[])
  {
   for (int k=1; k<=4; k++)
     {myprocess p = new myprocess(k,s1); //pass id and shared obj(s1)
      p.start();
     }
  }
}//main class: sample

class some //for shared object (monitor)
{
  private int counter = 1; //shared counter

  public synchronized void increment(int id)
  //synchronized method cannot be interrupted
  //only one thread can access at a time
  {
   System.out.println("process-"+id+" is Incrementing counter");
   System.out.println("--before counter= "+counter);
   counter++;
   System.out.println("--after counter= "+counter);
  }

  public synchronized void decrement(int id)
  //synchronized method cannot be interrupted
  //only one thread can access at a time
  {
   System.out.println("process-"+id+" is Decrementing counter");
   System.out.println("--before counter= "+counter);
   counter--;
   System.out.println("--after counter= "+counter);
  }
 }//class some

class myprocess extends Thread
{
  static some s1;
  private int id;
                      p id        shared_obj
  public myprocess(int k, some s1) //constructor
  { this.s1 = s1;
    id = k;
    System.out.println("===== Thread for process-"+id+ " created");
   }
```



```
   public void run()
   {
     try {sleep((int)(Math.random() * 2000));
          s1.increment(id);}
     catch (InterruptedException e)
         {System.out.println("Exception " + e.getMessage());}

     try {sleep((int)(Math.random() * 2000));
          s1.decrement(id);}
     catch (InterruptedException e)
         {System.out.println("Exception " + e.getMessage());}

     System.out.println("---- process-"+id+" terminates");
    }
}//class myprocess


//================================================================

     //for using "wait" in your program:
     //   try {wait();}
     //   catch (InterruptedException e)
     //       {System.out.println("Exception " + e.getMessage());}
     //
     //for using signal:
     //   notify(); --also try and catch is safer way
     //
```

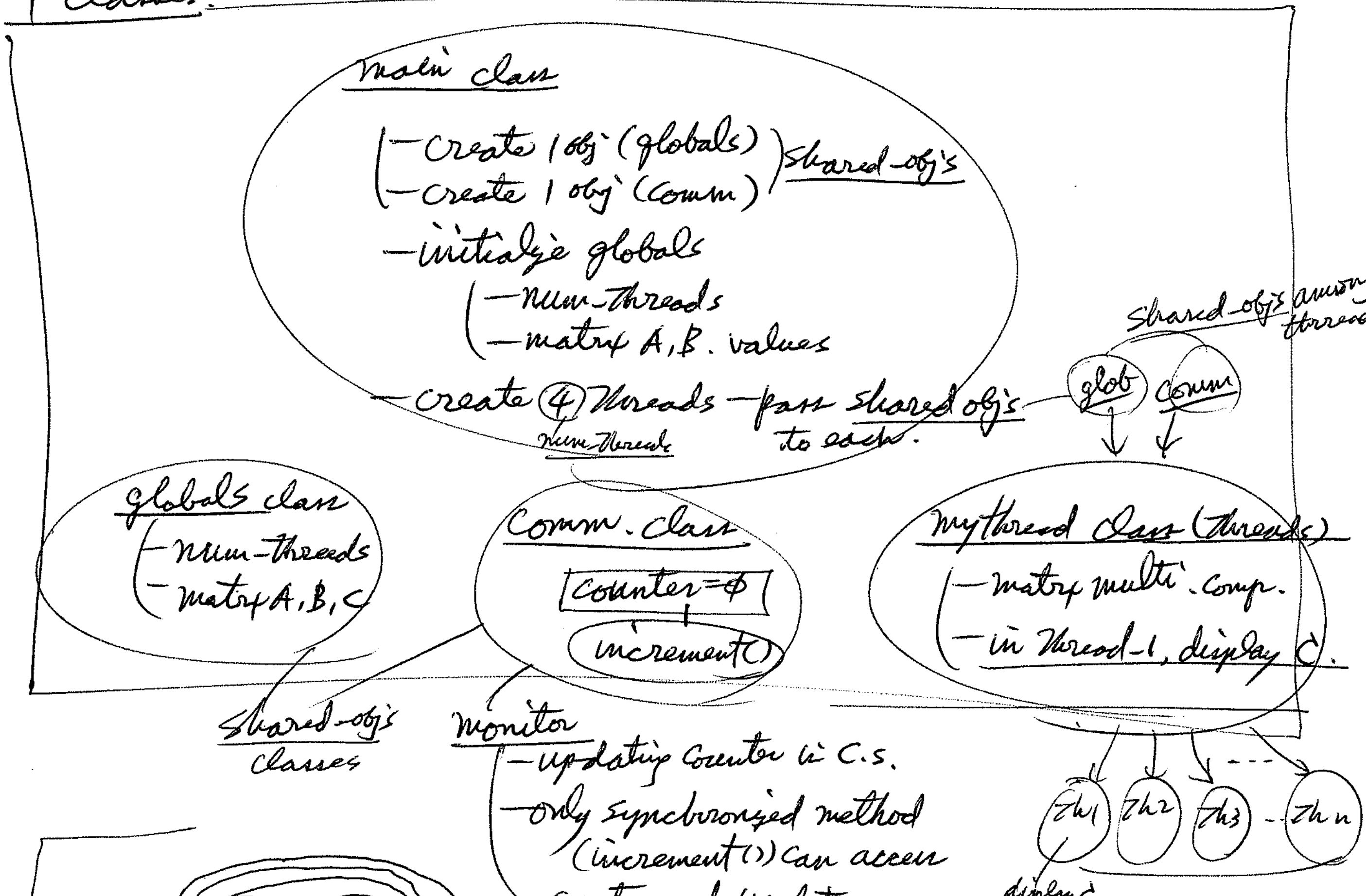
run 1

```
===== Thread for process-1 created
===== Thread for process-2 created
===== Thread for process-3 created
===== Thread for process-4 created
process-1 is Incrementing counter
--before counter= 1
--after counter= 2
process-3 is Incrementing counter
--before counter= 2
--after counter= 3
process-2 is Incrementing counter
--before counter= 3
--after counter= 4
process-2 is Decrementing counter
--before counter= 4
--after counter= 3
---- process-2 terminates
process-1 is Decrementing counter
--before counter= 3
--after counter= 2
---- process-1 terminates
process-4 is Incrementing counter
--before counter= 2
--after counter= 3
process-3 is Decrementing counter
--before counter= 3
--after counter= 2
---- process-3 terminates
process-4 is Decrementing counter
--before counter= 2
--after counter= 1
---- process-4 terminates
```

prg3 — Java Threads

- matrix-multi: A*B ⇒ C
- Comm/Synchronization — using shared object

4 classes:



main class
- create 1 obj (globals)
- create 1 obj (comm)

shared-obj's

- initialize globals
  - num-threads
  - matrix A, B. values
- create ④ threads — pass shared obj's to each.

num-threads

globals class
- num-threads
- matrix A, B, C

Comm. class

$\boxed{\text{counter} = \phi}$

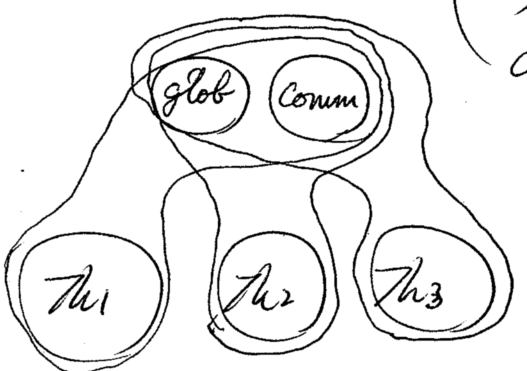
increment()

mythread class (threads)
- matrix multi. comp.
- in thread-1, display C.

shared-obj's classes

monitor
- updating counter in C.S.
- only synchronized method (increment()) can access counter and update.

display C (wait until all done, then display C :)

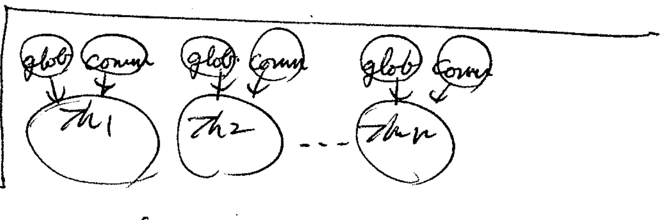


- We can display the resulting C matrix at the end of main process? ⇒ No!

  So, ~~[illegible]~~ make Th_1 do so.

- Threads share glob. and comm

  glob. contains A, B, C matrixes; comm having counter



Implementation:

When creating each thread, pass glob. and comm. (shared_obj's) as parameter (to constructor).