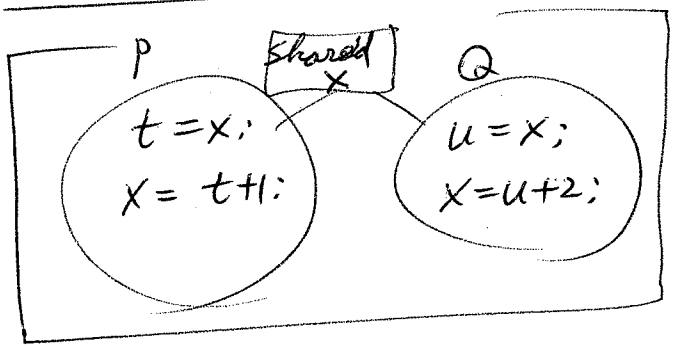
Concurrent programming

process (serial execution within process)

```
x = x+1;
x = x+2;
```

→ increments X ⇒ (X+3)

Concurrent processes (P, Q)

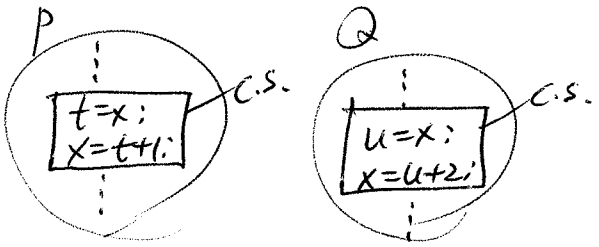


One possible interleaving:

```
t = x;
u = x;
x = u+2;
x = t+1
```

⇒ X ⇒ (X+1) ≠ incorrect

— Such interleaving can be prevented by treating the assignments as critical section (C.S.)



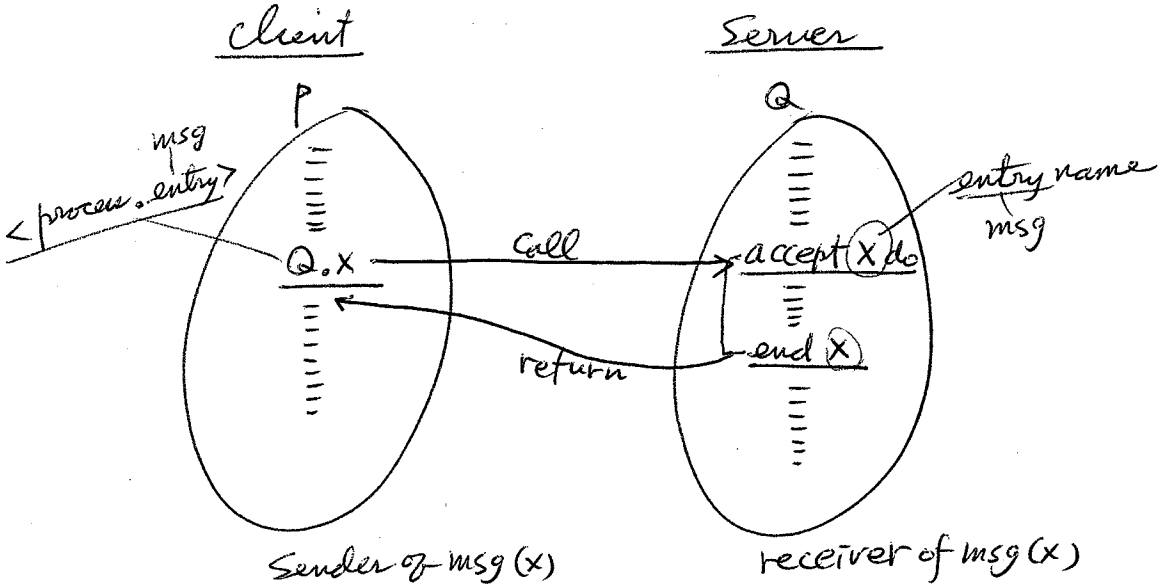
mutual exclusion.
(once started, no interruption until finish.

— Issues { Communication — { msg passing — Ada
shared-mem — Java
Synchronization

Ada { Communication — msg passing
Synchro. — Rendezvous

Java — shared-var
monitors — synchronized method (wait, notify)

Ada Rendezvous

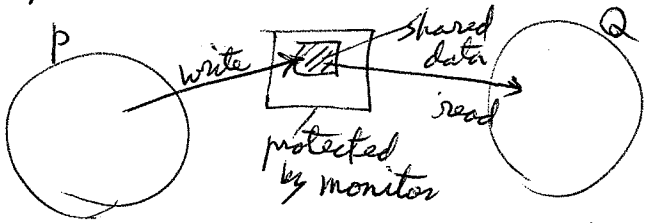


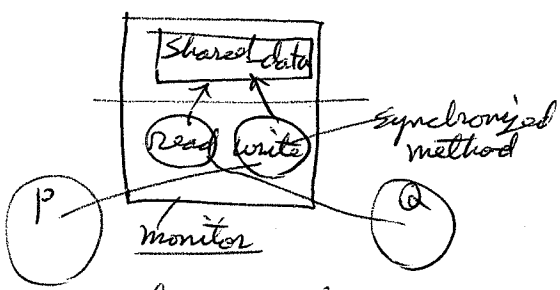
(Communication by msg passing
(Synchronization using Rendezvous

- if P calls (send msg) before Q is ready to accept, P waits until Rendezvous can occur.
- if Q reaches an accept statement before P calls, Q waits until Rendezvous can occur.

Java Threads

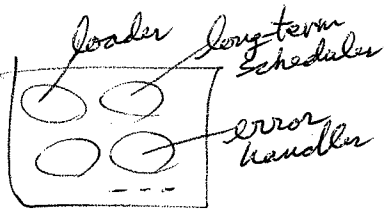
(Communication by shared data
(Synchronization by synchronized methods and wait/notify — monitor mechanism





(only 1 process (Th) can access synchronized method at a time.

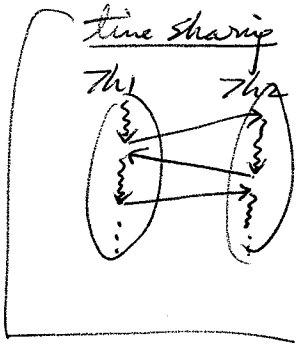
- Concurrent processes
  - op system kernels



  - User program (multi-threaded, multi-processed)
  - garbage collector

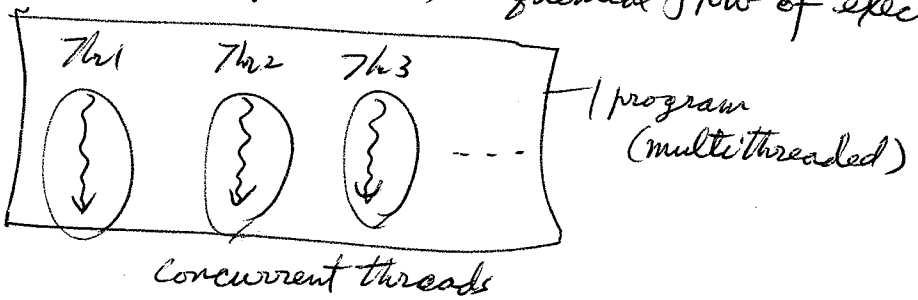
on uniprocessor
- time-sharing

on multi-processors
- parallel processing



- Why users write multi-threaded (multi-processed) program?
  - for high speed execution
  - for load sharing

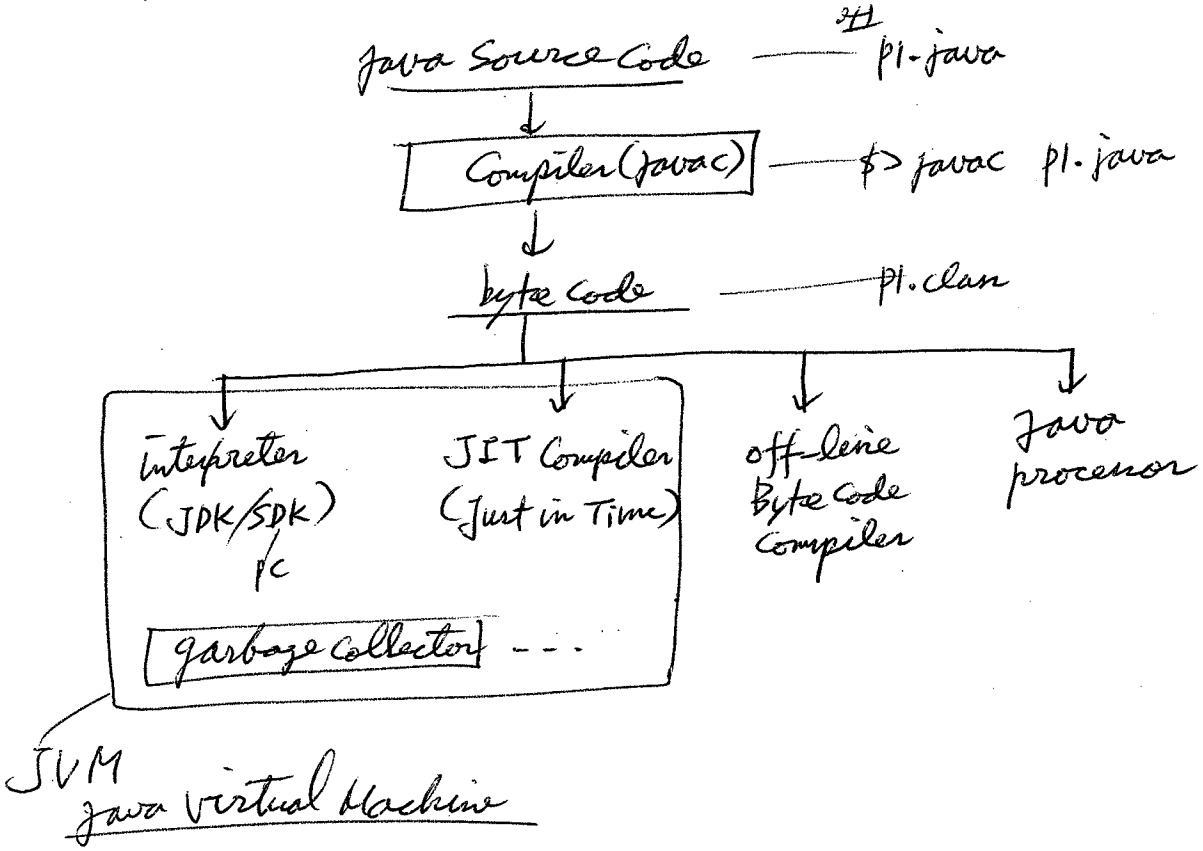
- What is a thread?
  - Thread is a single sequence of executable statements within a program.
  - within a single thread, sequential flow of execution.

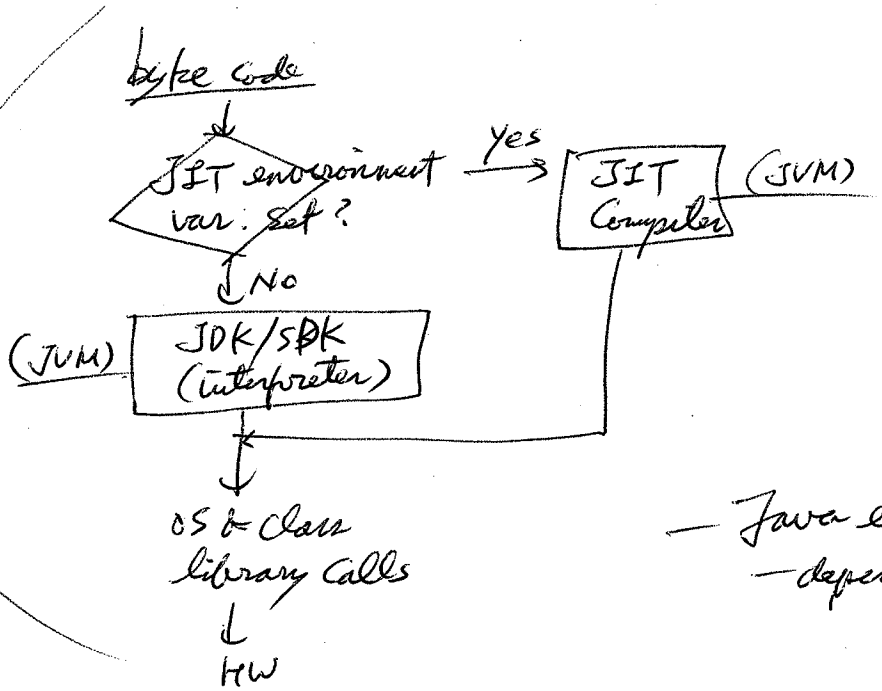


- JVM itself is a multi-threaded program.

— Java Threads — built-in support for parallel/concurrent computing.

@ — Java execution models

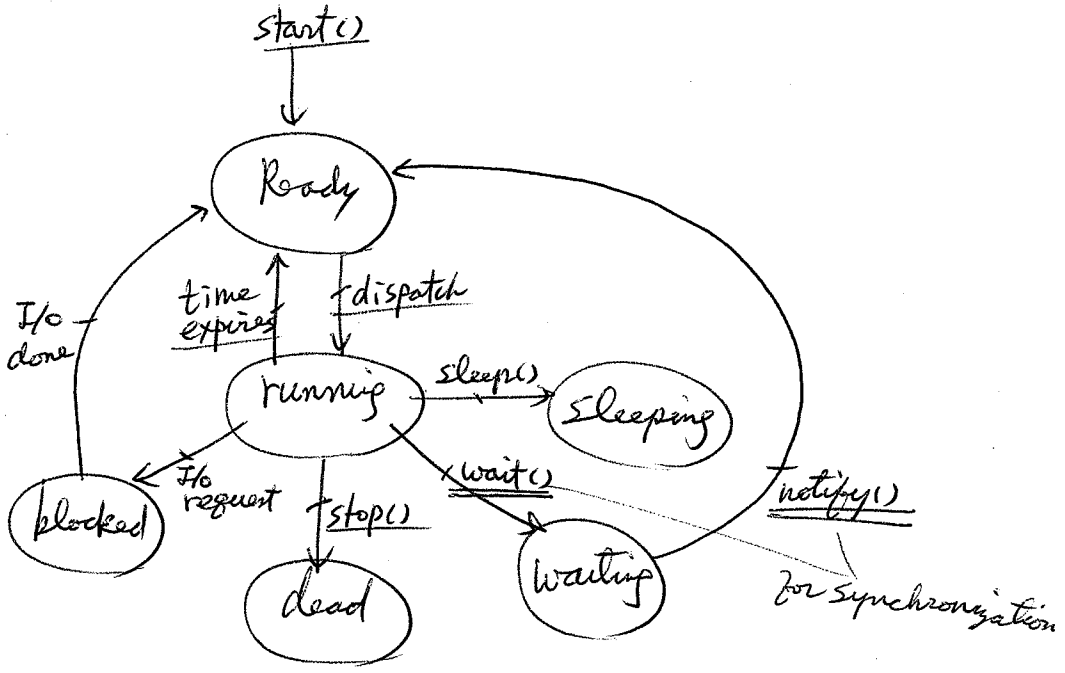




— Java execution tech. — depends on platforms

Java

— Life cycle of a thread



— How to write multithreaded programs

— [synchronize / communicate] threads

for proper order (monitor mechanism)

for cooperation (via shared data)

— on i5 — (multicore)

```
// test program for Java Threads
// 1 file version – only main class is public;
// File name must be the main class name, i.e., "Numbers.java"

public class Numbers
{ 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
```

(invokes run())

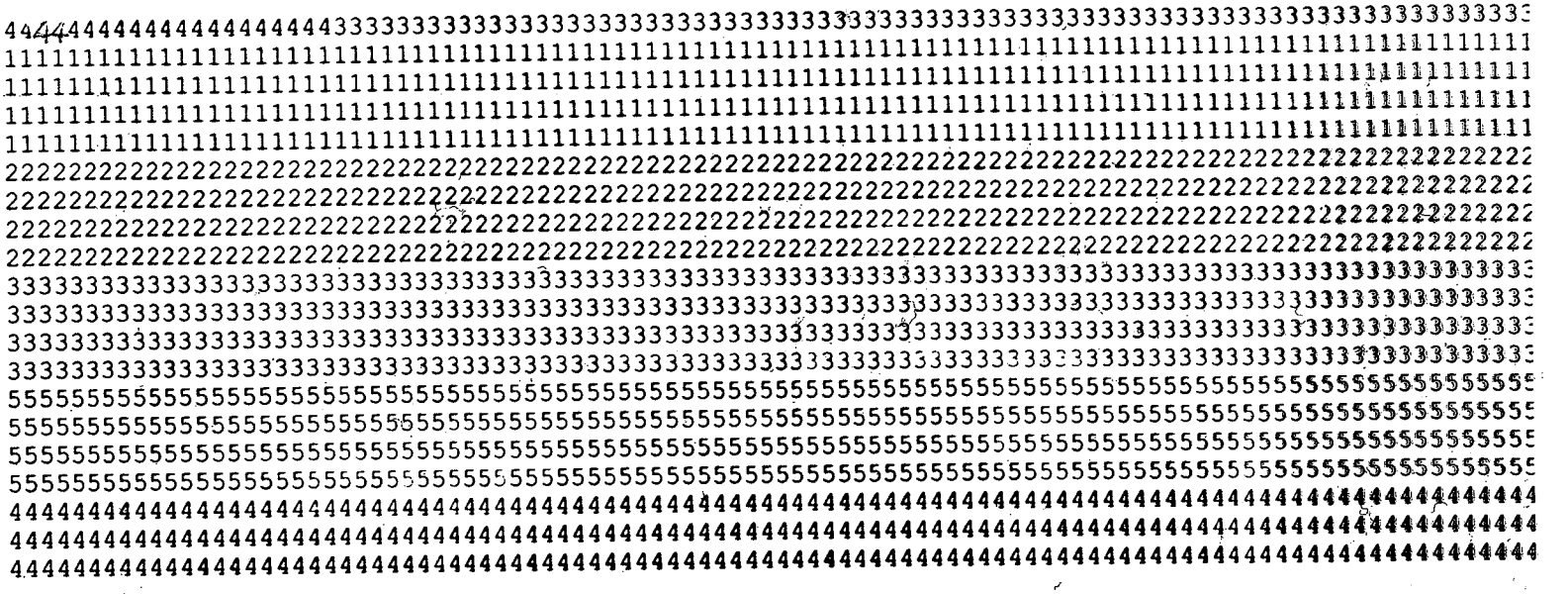
```
jpark$ javac Numbers.java
jpark$ java Numbers
11111111122222222222444444444411115533335511111111111111111111111422222222
22222222441111111155555555555555555555553333551142222224411111111115533333333
33333333333333333333333333333333333333333333333333333333333555555555555551544
44444444444444444444444444444444444444444444444444444444444444444444444444444
44442222222222222222222222222222222222222222222222222222222222222222555555555
55555555555555555555555555555555555555555555511111111111111111111111111111111
11111111133333333333333333333333
jpark$


jpark$ java Numbers
11111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111122222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222233333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333444444444444
44444444444444444444444444444444444444444444444444444444444444444444444444444
44444444445555555555555555555555555555555555555555555555555555555555555555555
55555555555555555555555555555555
jpark$
```

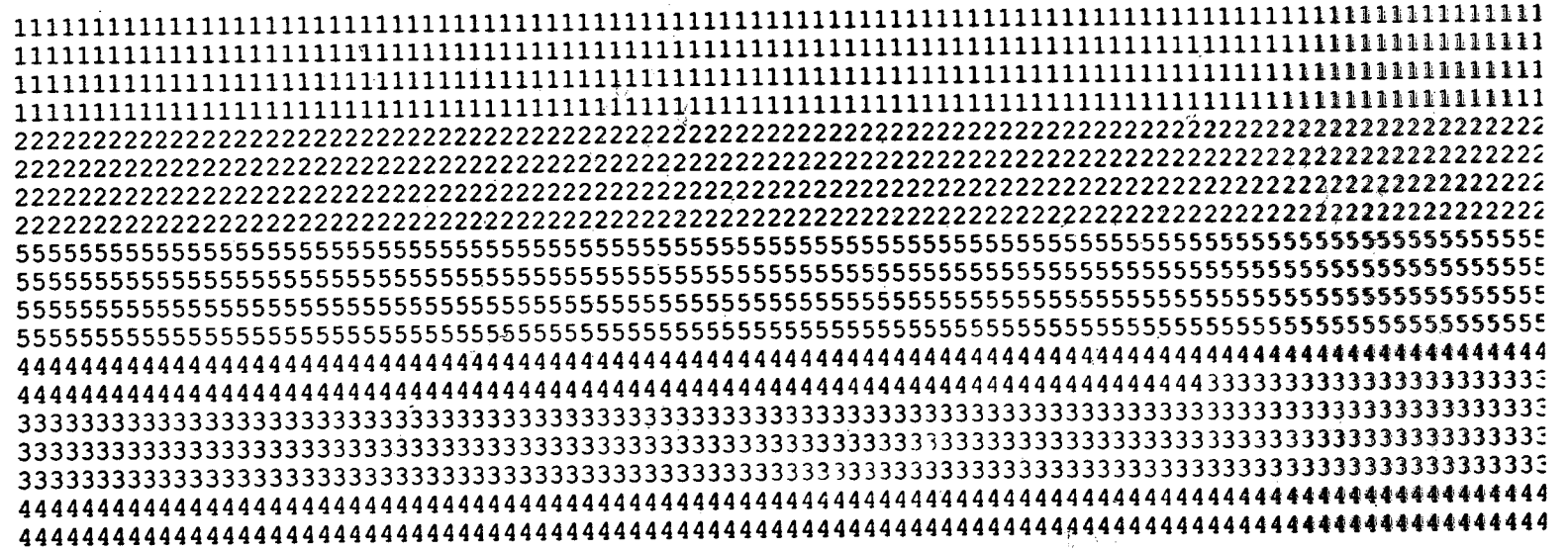
— on serial cpu —

```
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 (concurrent objects)

   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

11111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111122222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222223333333333
33333333333333333333333333333333333333333333333333333333333333333333
33333333333333333334444444444444444444444444444444444444444444444444
44444444444444444444444444444444444444444444444455555555555555555555
55555555555555555555555555555555555555555555555555555555555555555555
5555555555

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

—on serial cpu—

for (k=0; k<1000; k++) —longer threads





```
Script started
zodiac:~/JAVA/Num > cat Num2.java
//This is a test multithreaded program in which 5 threads are created
//and executed concurrently.
//using sleep(), each thread yields the turn.

public class Num2  //in a single file version, only the 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++)
    {
     try {Thread.sleep((int)(Math.random()*1000));}
     catch (InterruptedException e)
          {System.out.println(e.getMessage());}

     System.out.print(num);
     }
   }//run
 }//NumberThread

zodiac:~/JAVA/Num > java Num2

351214354255132314442544531253224144523145124532413542411532431151 4
123342415225531245341242231354122332254215541212124351425134323153 3
455214521235541223224135432124312453152423145343214253435314232514 5
351215541325412323452413525334243215421254325135441425455334412352 2
435151252415513525435312342251535412145322351452315453244535312531 3
141424532521344152435154323514223453241243521143453523513422353135 4
124532513142534225314251453241341245431245344251434241521434531124 1
534523133151535133113311111111 1
zodiac:~/JAVA/Num > exit
exit

script done
```

0~1

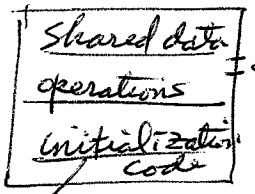
— sleep yields turn to other threads

Java Threads

- User program can be Single-Threaded or multi-Threaded.
- even if Single-Threaded, class libraries create threads to handle finalization and weak references.
- JVM may spawn threads for garbage collection, compilation, etc.

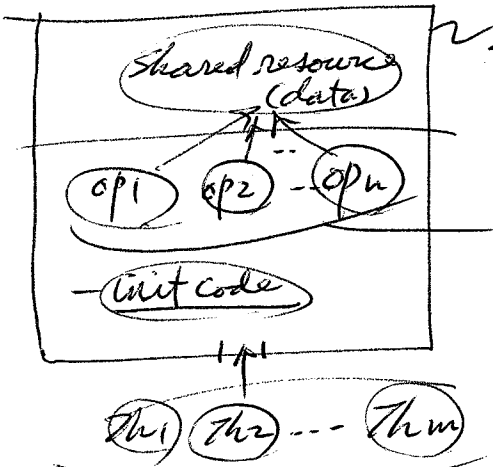
— Monitor Synchronization — used in Java Threads

monitor Concept:



- When monitor is activated, executed only once.
- only 1 process (thread) at a time is active in the monitor.
- Mutex is implicit

Java Threads



synchronized methods (in the shared resource class)
only 1 op can access shared-resource at a time. (∵ synchro. methods).
⇒ mutex

waiting threads on a shared-resource object.

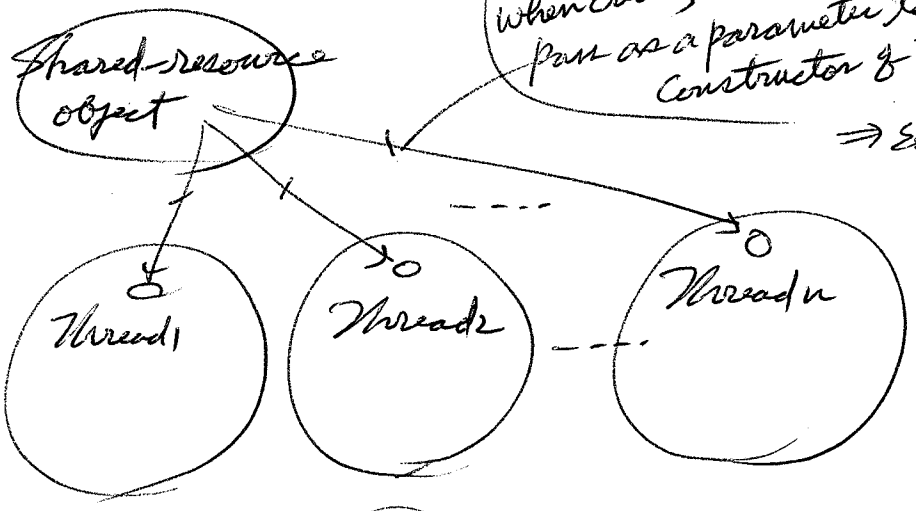
(1 Thread at a time can access shared data through a synchronized method.)

- monitor
  - is an abstract data type (higher-level construct than semaphore)
  - encapsulates shared data and operations.

— Synchronizing Java threads

- Synchronized methods in the shared resource class
  - uses [wait() / notify()]
  - (monitor mechanism)
- multiple threads can call synchronized methods in the shared-resource object, but only 1 thread can get access at a time.

— implementing shared data among multiple threads.



eg) Shared-data-class shrobj;
Threadclass Th1, Th2, Th3, Th4; // 4 threads in Threadclan (objects)

[Th1 = new Threadclan(©); — create a thread
Th1.start(); — starts the thread.

[Th2 = new Threadclan(©);
Th2.start();
---

eg) class Threadclan extends Thread
{
constructor (making a class as a Thread.
(Shared-data-class shrobj) {...}
input parameter
}

— Serially Reusable Resources problem — monitor solution

```
monitor SR
    export: acquire, release;
    int avail; [4]
    (condition) event ready;
    [ begin
        avail = 4;      ) — init code.
      end;
```

— only 1 process can enter monitor.
⇒ only 1 of the 2 functions is activated at a time.

```
function (operation) acquire
    if (avail == 0)
    → wait (ready);      — process locked on "ready".
    avail = avail - 1;

function (operation) release
    avail = avail + 1;
    if (avail == 1)
        signal (ready);
```

signal awakens 1 of waiting processes on "ready".

a releasing process unlocks 1 of waiting processes on the same condition code ("ready").