LAB 3: NETWORK DISTRIBUTED SYSTEMS

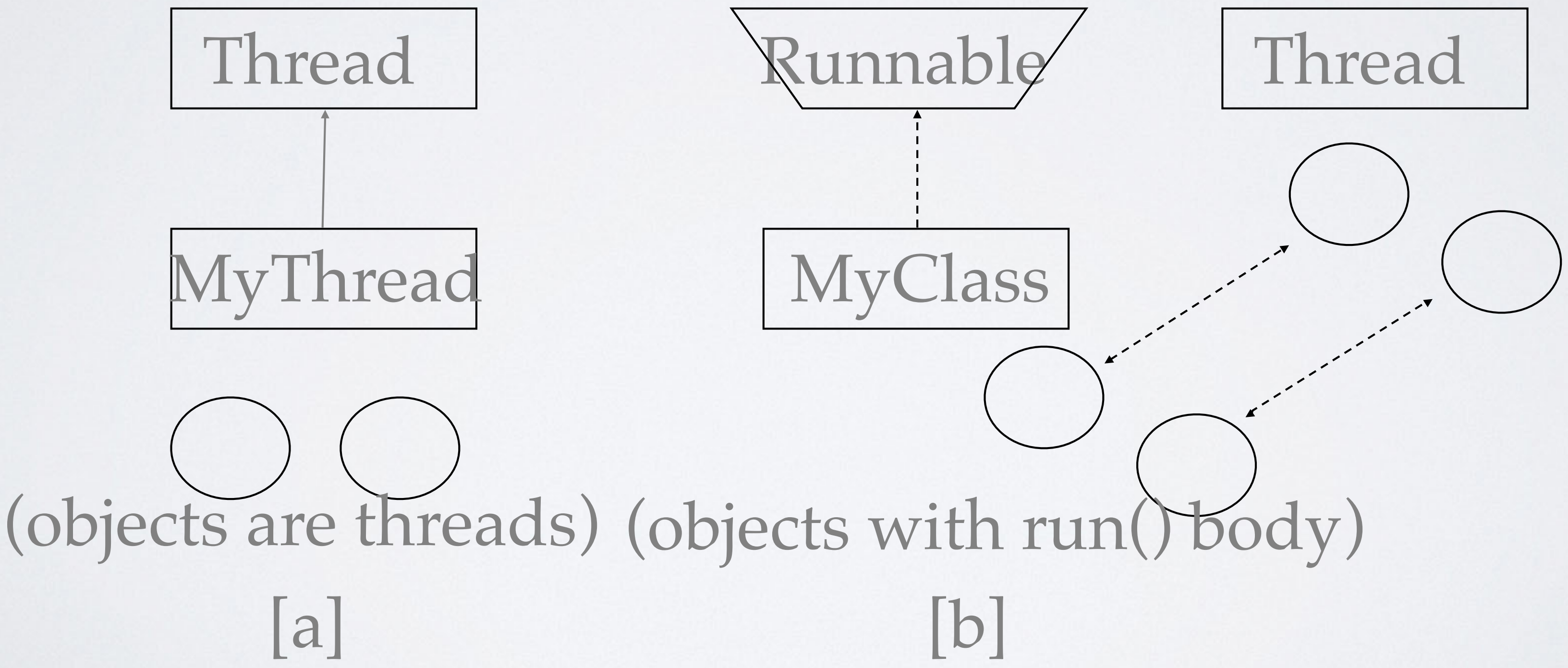
Java Multithreading

JAVA THREADS

- Java has built in support for Multithreading
- Synchronization
- Thread Scheduling
- Inter-Thread Communication:
  - currentThread start setPriority
  - yield run getPriority
  - sleep stop suspend
  - resume
- Java Garbage Collector is a low-priority thread.

THREADING MECHANISMS...

- Create a class that extends the Thread class
- Create a class that implements the Runnable interface



1ST METHOD: EXTENDING THREAD CLASS

- Create a class by extending Thread class and override run() method:

```
class MyThread extends Thread
{
    public void run()
  {
     // thread body of execution
  }
 }
```

- Create a thread:
- `MyThread thr1 = new MyThread();`
- Start Execution of threads:
- `thr1.start();`
- Create and Execute:
- `new MyThread().start();`

AN EXAMPLE

```
class MyThread extends Thread {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx1 {
        public static void main(String [] args  ) {
       MyThread t = new MyThread();
        t.start();
        }
}
```

2ND METHOD: THREADS BY IMPLEMENTING RUNNABLE INTERFACE

- Create a class that implements the interface Runnable and override run() method:

```
class MyThread implements Runnable
{
   .....
   public void run()
   {
      // thread body of execution
   }
}
```

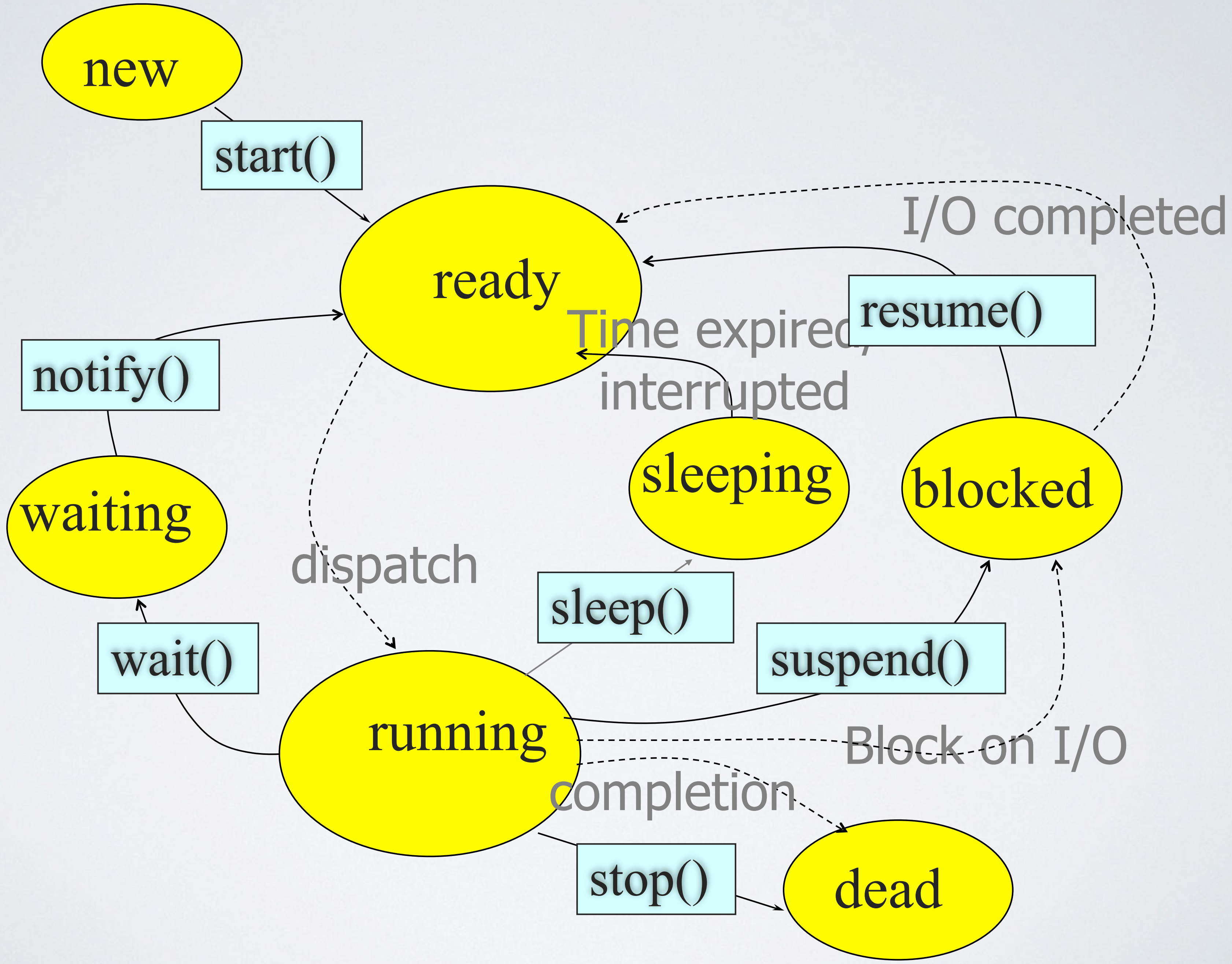
- Creating Object:
- `MyThread myObject = new MyThread();`
- Creating Thread Object:
- `Thread thr1 = new Thread( myObject );`
- Start Execution:
- `thr1.start();`

AN EXAMPLE

```
class MyThread implements Runnable  {
        public void run() {
                System.out.println(" this thread is running ... ");
        }
}

class ThreadEx2 {
        public static void main(String [] args  ) {
                Thread t = new Thread(new MyThread());
                 t.start();
        }
}
```

LIFE CYCLE OF THREAD



A PROGRAM WITH THREE JAVA THREADS

- Write a program that creates 3 threads

THREE THREADS EXAMPLE

```
class A extends Thread
{
       public void run()
         {
              for(int i=1;i<=5;i++)
                {
                      System.out.println("\t From ThreadA: i= "+i);
                }
                 System.out.println("Exit from A");
         }
}

class B extends Thread
{
       public void run()
         {
              for(int j=1;j<=5;j++)
                {
                      System.out.println("\t From ThreadB: j= "+j);
                }
                 System.out.println("Exit from B");
         }
}
```

THREE THREADS EXAMPLE

```
class C extends Thread
{
       public void run()
         {
              for(int k=1;k<=5;k++)
                {
                      System.out.println("\t From ThreadC: k= "+k);
                }

                 System.out.println("Exit from C");
         }
}

class ThreadTest
{
          public static void main(String args[])
           {
                    new A().start();
                    new B().start();
                    new C().start();
           }
}
```

RUN 1

```
[raj@mundroo] threads [1:76] java ThreadTest
        From ThreadA: i= 1
        From ThreadA: i= 2
        From ThreadA: i= 3
        From ThreadA: i= 4
        From ThreadA: i= 5
Exit from A
        From ThreadC: k= 1
        From ThreadC: k= 2
        From ThreadC: k= 3
        From ThreadC: k= 4
        From ThreadC: k= 5
Exit from C
        From ThreadB: j= 1
        From ThreadB: j= 2
        From ThreadB: j= 3
        From ThreadB: j= 4
        From ThreadB: j= 5
Exit from B
```

RUN 2

```
[raj@mundroo] threads [1:77] java ThreadTest
    From ThreadA: i= 1
    From ThreadA: i= 2
    From ThreadA: i= 3
    From ThreadA: i= 4
    From ThreadA: i= 5
    From ThreadC: k= 1
    From ThreadC: k= 2
    From ThreadC: k= 3
    From ThreadC: k= 4
    From ThreadC: k= 5
Exit from C
    From ThreadB: j= 1
    From ThreadB: j= 2
    From ThreadB: j= 3
    From ThreadB: j= 4
    From ThreadB: j= 5
Exit from B
Exit from A
```

THREAD PRIORITY

- In Java, each thread is assigned priority, which affects the order in which it is scheduled for running. The threads so far had same default priority (NORM_PRIORITY) and they are served using FCFS policy.
  - Java allows users to change priority:
    - ThreadName.setPriority(intNumber)
      - MIN_PRIORITY = 1
      - NORM_PRIORITY=5
      - MAX_PRIORITY=10

THREAD PRIORITY EXAMPLE

```
class A extends Thread
{
        public void run()
          {
                System.out.println("Thread A started");
                for(int i=1;i<=4;i++)
                  {
                         System.out.println("\t From ThreadA: i= "+i);
                  }
                   System.out.println("Exit from A");
          }
}
class B extends Thread
{
        public void run()
          {
                System.out.println("Thread B started");
                for(int j=1;j<=4;j++)
                  {
                         System.out.println("\t From ThreadB: j= "+j);
                  }
                   System.out.println("Exit from B");
          }
}
```

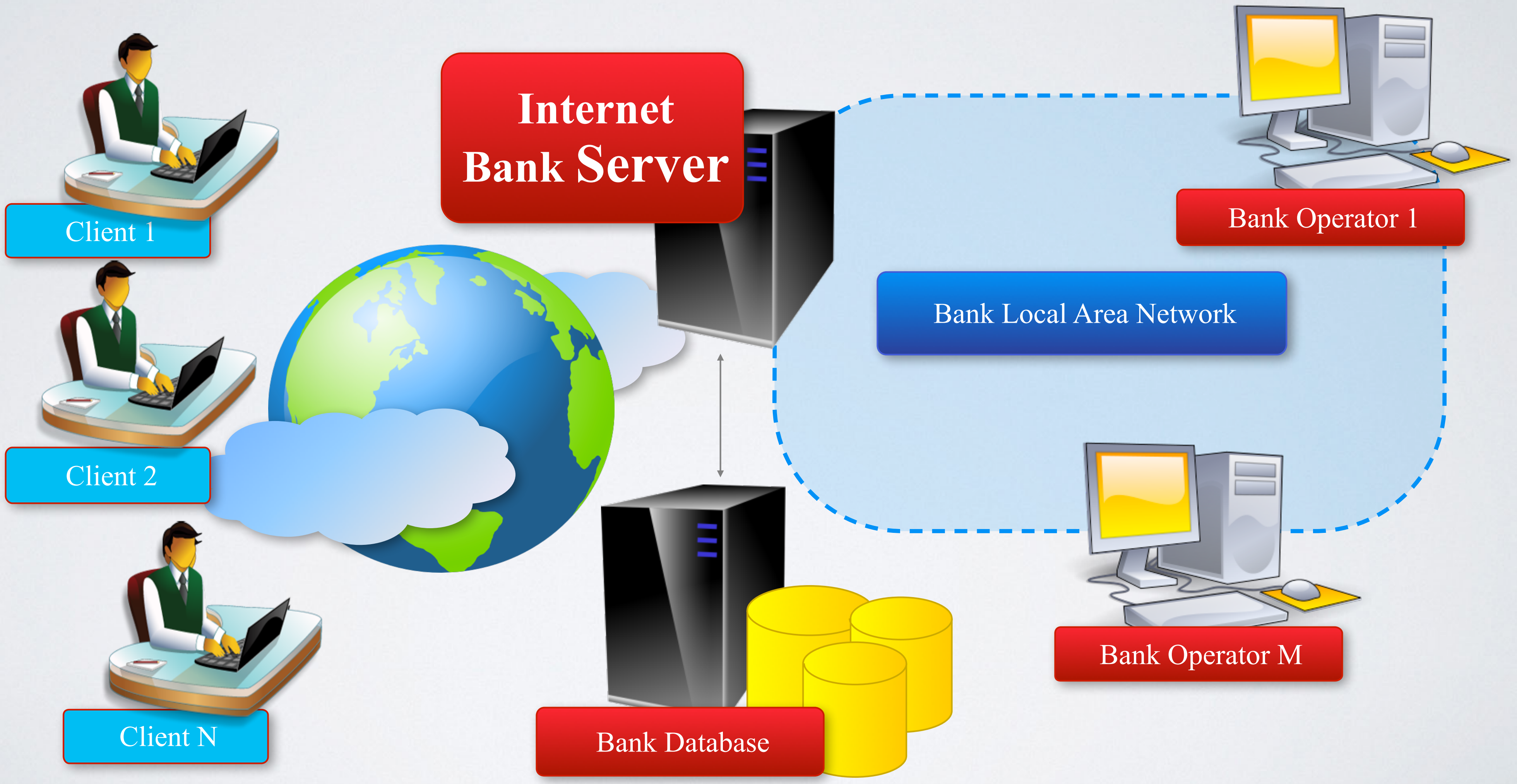
THREAD PRIORITY EXAMPLE

```
class C extends Thread
{
        public void run()
         {
               System.out.println("Thread C started");
               for(int k=1;k<=4;k++)
                 {
                        System.out.println("\t From ThreadC: k= "+k);
                 }
                 System.out.println("Exit from C");
         }
}
class ThreadPriority
{
           public static void main(String args[])
            {
                        A threadA=new A();
                        B threadB=new B();
                        C threadC=new C();
                      threadC.setPriority(Thread.MAX_PRIORITY);
                      threadB.setPriority(threadA.getPriority()+1);
                      threadA.setPriority(Thread.MIN_PRIORITY);
                      System.out.println("Started Thread A");
                       threadA.start();
                      System.out.println("Started Thread B");
                       threadB.start();
                      System.out.println("Started Thread C");
                       threadC.start();
                       System.out.println("End of main thread");
            }
}
```

ACCESSING SHARED RESOURCES

- Applications access to shared resources need to be coordinated.
  - Printer (two people's jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account.
  - Can the following operations be done at the same time on the same account?
    - Deposit()
    - Withdraw()
    - Enquire()

ONLINE BANK: SERVING MANY CUSTOMERS AND OPERATIONS



SHARED RESOURCES

- If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
- This can be prevented by synchronising access to the data.
- Use "Synchronized" method:
  - public synchronized void update()
  - {
    - …
  - }

THE DRIVER: 3 THREADS SHARING THE SAME OBJECT

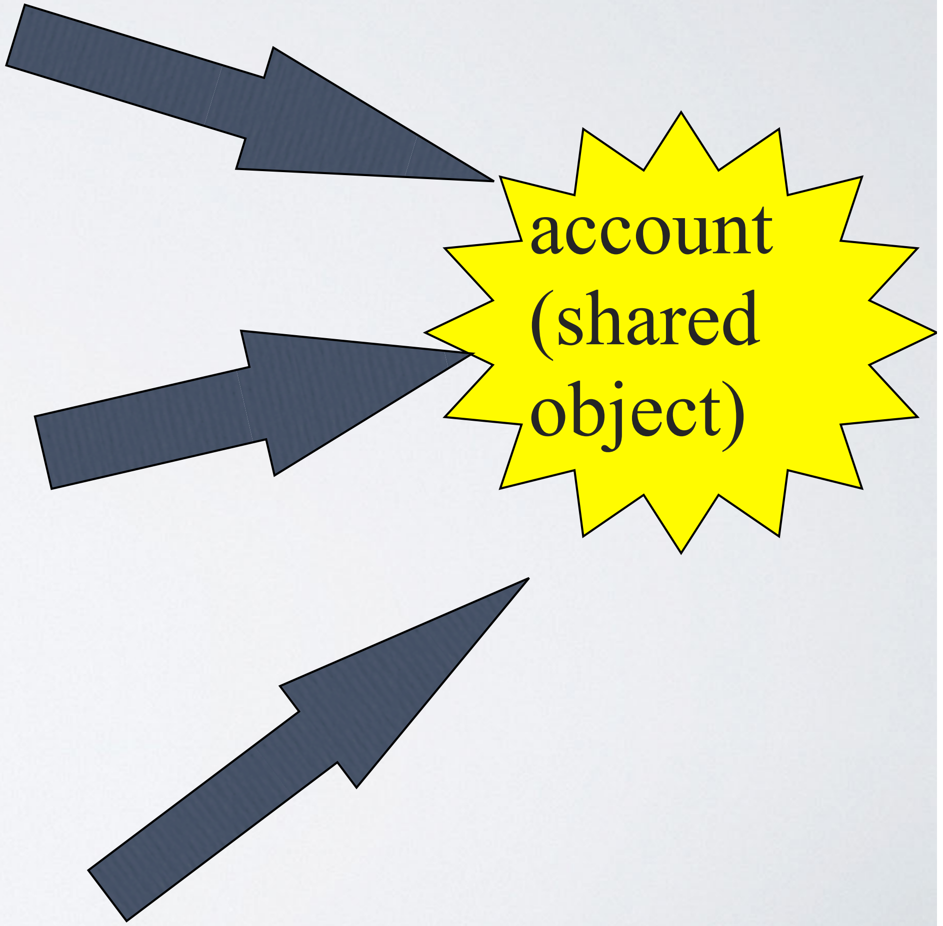
```
class InternetBankingSystem {
        public static void main(String [] args  ) {
            Account accountObject = new Account ();
            Thread t1 = new Thread(new MyThread(accountObject));
               Thread t2 = new Thread(new YourThread(accountObject));
               Thread t3 = new Thread(new HerThread(accountObject));
            t1.start();
            t2.start();
            t3.start();
          // DO some other operation
        } // end main()
}
```

SHARED ACCOUNT OBJECT BETWEEN 3 THREADS

```
class MyThread implements Runnable  {
 Account account;
        public MyThread (Account s) {  account = s;}
        public void run() { account.deposit(); }
} // end class MyThread

class YourThread implements Runnable  {
 Account account;
        public YourThread (Account s) { account = s;}
        public void run() { account.withdraw(); }
} // end class YourThread

class HerThread implements Runnable  {
 Account account;
        public HerThread (Account s) { account = s; }
        public void run() {account.enquire(); }
} // end class HerThread
```



MONITOR (SHARED OBJECT ACCESS): SERIALIZES OPERATION ON SHARED OBJECTS

```
class Account {   // the 'monitor'
   int balance;

      // if 'synchronized' is removed, the outcome is unpredictable
       public synchronized void deposit( ) {
          // METHOD BODY : balance += deposit_amount;
        }

         public synchronized void withdraw( ) {
          // METHOD BODY: balance -= deposit_amount;
         }
         public synchronized void enquire( ) {
          // METHOD BODY: display balance.
         }
}
```