new Runnable() is for creating object which implements the runnable interface but no class name right?
  Ans. Yes (Ananomyos implementation)

notify have inbuilt logic to inform to the exact thread which release the lock?
   notifyAll does not have this?
notify() -> afte the updation is done it would just notify to the waiting thread

```
                  eg:
                              obj1.wait()(1 thread)
                                    |
                                    | ==========> obj1.notify()
                                    |
                               waiting state
```

notifyAll() -> after the updating is done it will notify all the waiting threads

```
                         obj1.wait() =====> 30 threads
                        obj2.wait() =====> 60  threads

                        obj1.notifyAll()---> notification will be sent to 30
threads
```

Q>

```
psvm(Striing[] args){
A a = new A();
B b = new B();
 Thread t1 = new Thread(a);
 Thread t2 = new Thread(b);
      t1.start();
      t2.start();
}
```

Sir here will the main method immediately execute t1.start() and t2.start()?
Will the thread scheduler allocate cpu for t1 thread, will make t1 in running and main thread in runnable
(in the line t1.start())?
Could you please explain that sir?

3 threads
      a. main thread
      b. t1 thread
      c. t2 thread

Q>
when we have sb.notify() how will will waiting thread know from which point of code to start?
      sb.notify() and sb.wait() both are under synchronized and they share the common communication channel.

```
            sb.wait() -> release the lock and enter into waiting state
                  |
            sb.notify() -> use the lock update the object and send the notification
to sb.wait()
```

can you please tell the difference between join and yield?
    join() -> stop ur execution and wait for another thread to execute later join
   yield() -> pause ur execution for sometime and check whether there are any

threads of same priority available or
not, if not continue ur execution, if available give the other thread a chance.

so how we are assuming consumer will take the lock , it is like lock will not be given to producer and given to
consumer first?

```
    Producer ->produce the item and sent the notification to the consumer[notify()]
    Consumer -> consume the item if avaialble otherwise wait as long as the producer produces the item
                        and gets the notification.[wait]

     consumer.start();//no enter into waiting state by releasing the lock
     producer.start()// get the lock and update the data and finally relase the lock.
```

Q>
main thread -> trigger logic[t2.start(),t1.start()]
producer thread -> notify()[produces the item and gives the notification for waiting thread expecting the lock]
consumer thread -> wait()[lock relased and waiting for notification]

Q>

```
class ThreadB extends Thread{
            int total =0;
            public void run(){
                        for(;;;)
                              update total;
            }
}
class Test{
      p.s.v.m(String... args){
            ThreadB b=new ThreadB();
            b.start();

            System.out.println(b.total);
      }
}
main thread[5] ->  System.out.println(b.total) //0
user thread[5]->
      int total =0;
      for(;;)
            update total
```

Q>

```
main(String.. args){
      Thread t1=new Thred();
      Thread t2 =new Thread();

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

}
3 threads
     main thread(currently executing)
      t1
      t2[ got  a chance]
```

```
Q>
     t1 -----> ;;;;;;;;;;;;;Thread.sleep(100);;;;;;;;;;
     t2 -----> t1.join()[waiting state]
     t3------> ;;;;;;;;;;;;;;;;Thread.sleep(4000);;;;;;;

Q>
class Thread{
      public void start(){
            //register the thread with ThreadScheduler
            // perofm low level memeory management
            //makes a call to run()
      }
}
class Test extends Thread{
            public void start(){
                  super().start();
            }
}

SharedResource usage in effiecient way without datainconsistency(synchronized
region)

t1[5]---> b.wait()
t2[5]--->;;;;;;; b.notify()
```