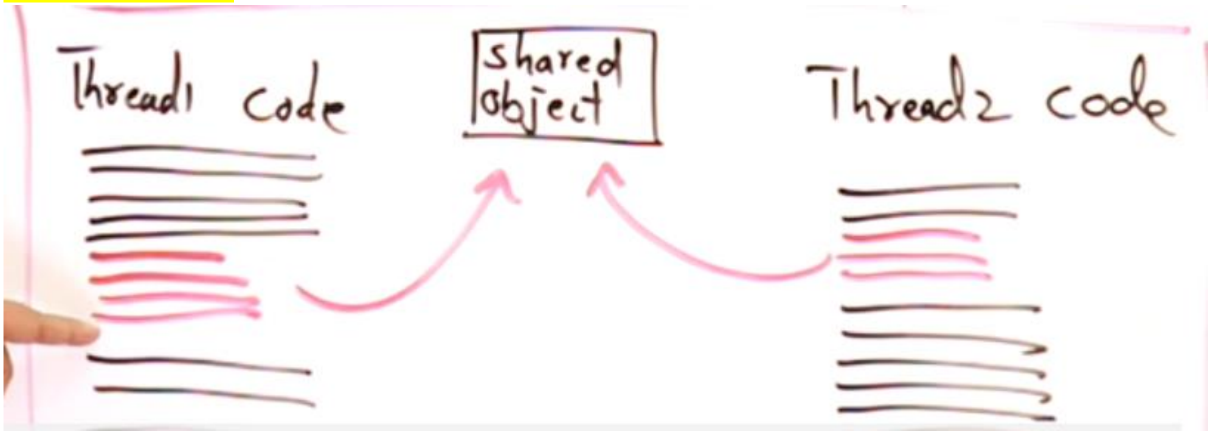
Synchronization

All threads will have their individual stack. But object is created in heap.

Resource sharing

Critical Section



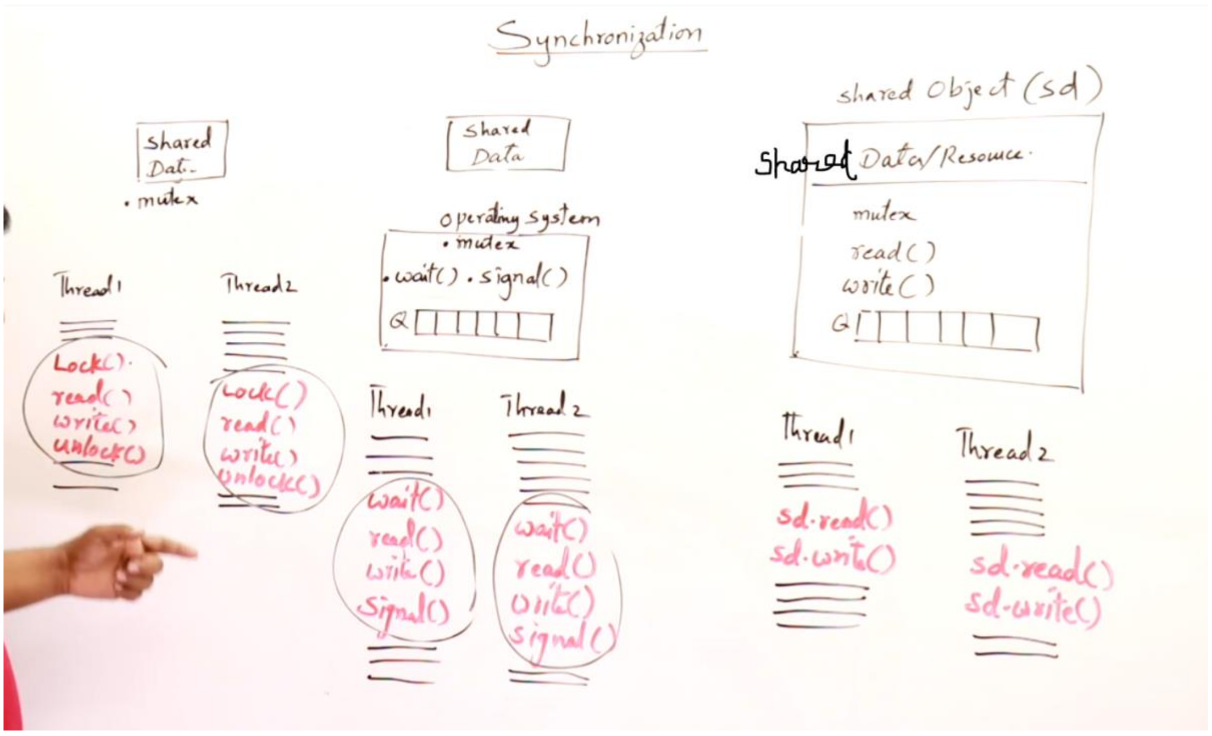
Red lines are the critical section part of their respective codes.

Mutual Exclusion

Will not allow two threads to access the same resource simultaneously. This is known as Mutual exclusion. Preventing any other thread to access the same object is known as mutual exclusion. There should be some co-ordination between threads so that one thread can access the shared object at that instant.

Happening of one prevents the happening of other.
Thread-1 accessing a resource will prevent thread-2 to access.

1) There should be some system which should take care of the shared resources by allowing only 1 thread at a time → Locking/mutex || semaphores || monitor.
   Locking/mutex (mutex will take care)
   Semaphores (OS will take care)
   Monitor (objects will take care)
2) Apart from the system there must be some co-ordination between threads (i.e t-1 and t-2 should communicate with each-other to access the shared resource. → Race condition || Inter-thread communication.



Locking/Mutex Semaphore Monitor

Locking/Mutex

Here threads will lock() , read() , write() , unlock() automatically. Two / more threads will communicate with each-other and provide mutual exclusion.

Semaphore

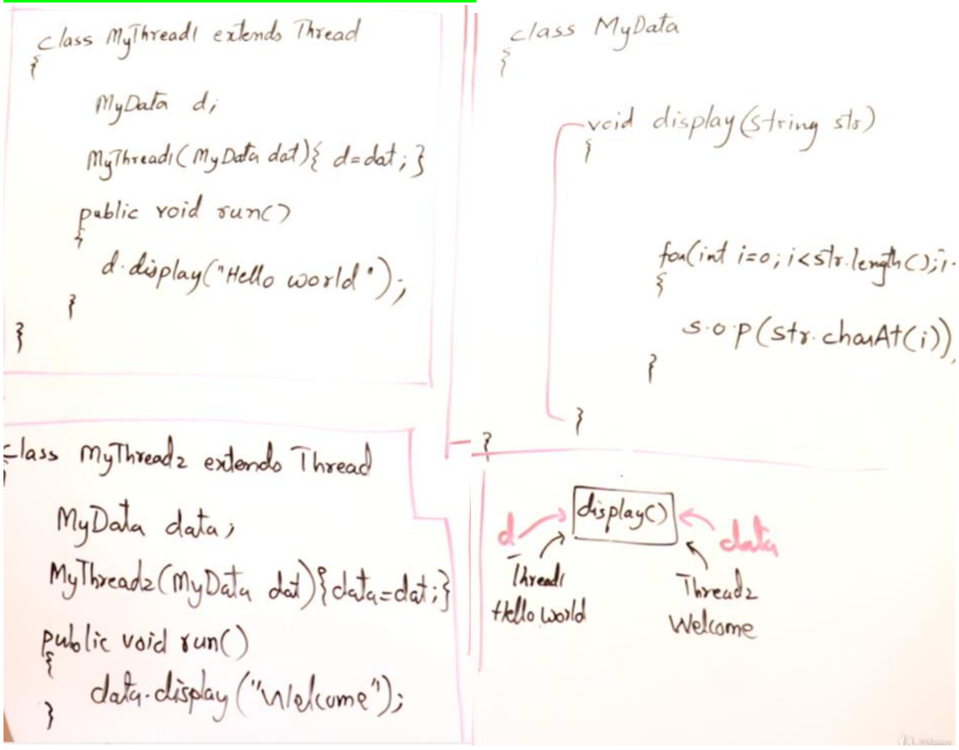
Threads knows how to read and write, OS will take care of locking [i.e wait()] and unlock [i.e signal()]. Operating System will achieve mutual exclusion.

Monitor

Thread doesn't know how to read and write. But it can perform by calling the methods. Objects in Object Oriented Programming(OOPS) will contain data and methods() and these objects provides mutual exclusion.

Eg: Barber Shop, Customers wants hair-cut, trim bears but customers don't know how to do??
So they simply calls the methods which they want, and stylists will make the customers sit in the queue and allow one-by-one when their turn comes.

Multi-threading using Monitor



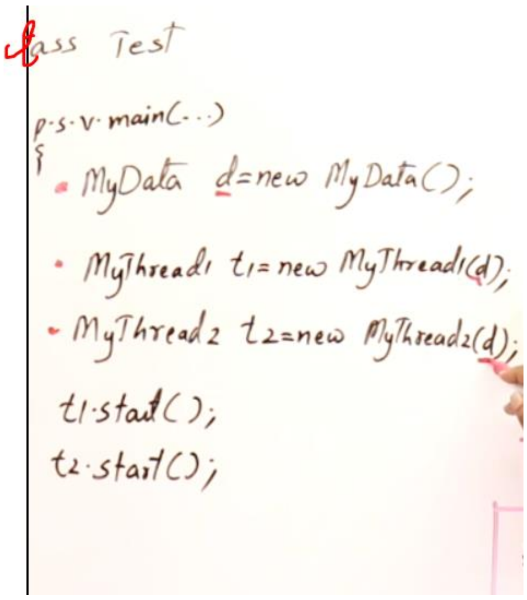
```
class MyThread1 extends Thread
{
    MyData d;
    MyThread1(MyData dat){ d=dat; }
    public void run()
    {
      d.display("Hello world");
    }
}
```

```
class MyData
{
   void display(String str)
   {
        for(int i=0;i<str.length();i
        {
          s.o.p(str.charAt(i))
        }
   }
}
```

```
class MyThread2 extends Thread
{
   MyData data;
   MyThread2(MyData dat){data=dat;}
   public void run()
   {
      data.display("Welcome");
   }
```

d
display()
data
Thread1
Hello world
Thread2
Welcome

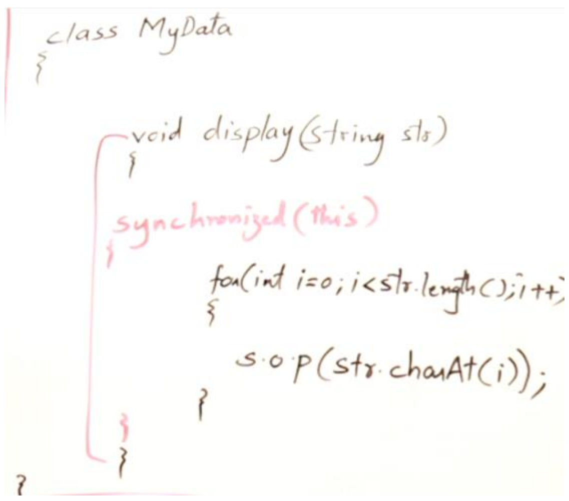
Main method



```
class Test

p.s.v.main(...)
{
 • MyData d=new MyData();
 • MyThread1 t1= new MyThread1(d);
 • MyThread2 t2=new MyThread2(d);
   t1.start();
   t2.start();
```

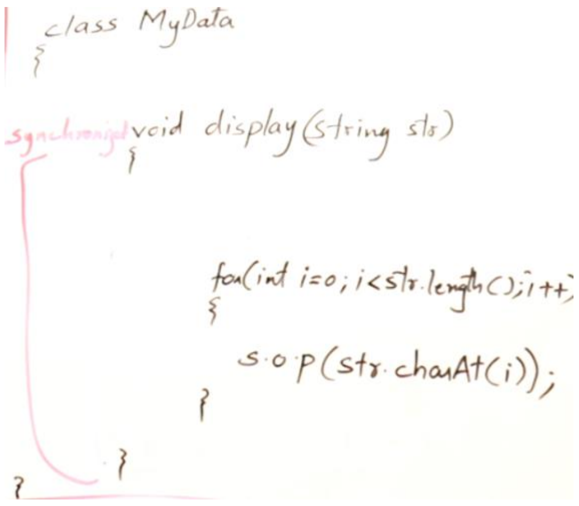
Here d is the shared object to all the threads.

Method-1

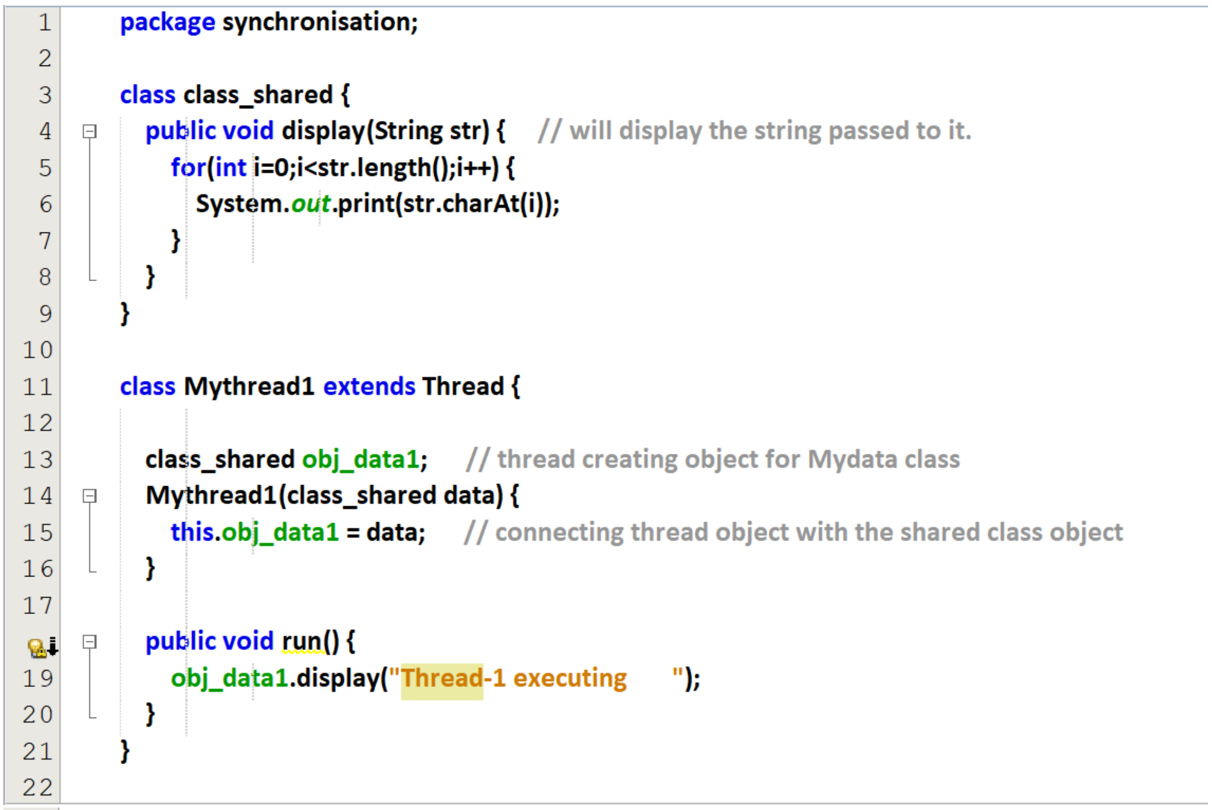


```
class MyData
{
    void display(String str)
    {
    synchronized(this)
    {
        for(int i=0;i<str.length();i++
        {
            s.o.p(str.charAt(i));
        }
    }
    }
}
```

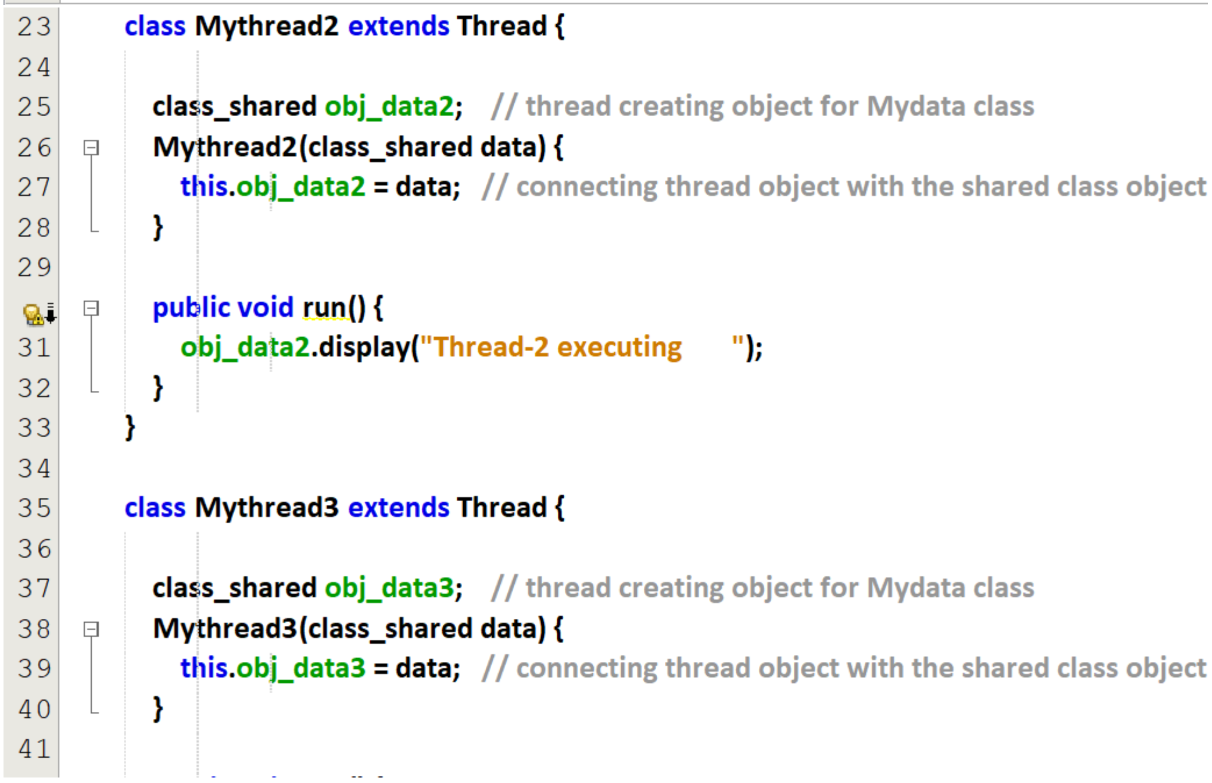
Method-2



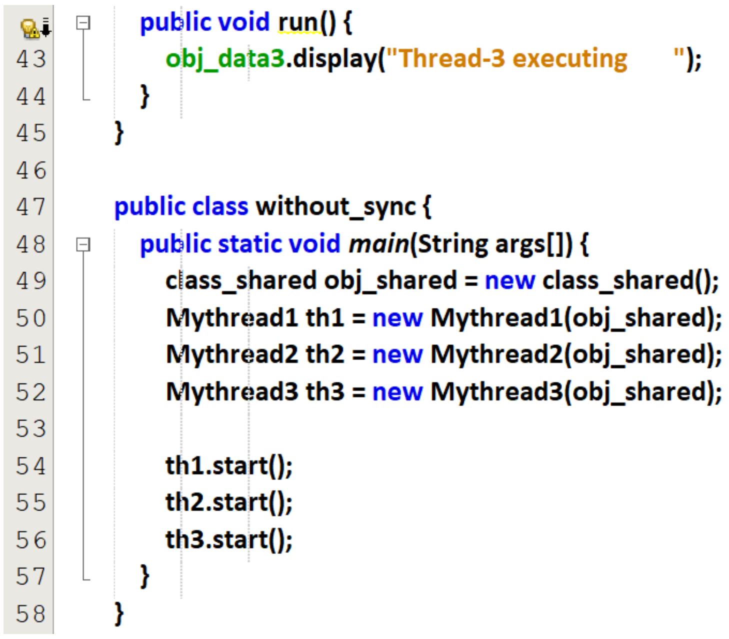
```
class MyData
{
synchronized void display(String str)
    {


        for(int i=0;i<str.length();i++
        {
            s.o.p(str.charAt(i));
        }

    }
}
```



```java
package synchronisation;

class class_shared {
    public void display(String str) {    // will display the string passed to it.
        for(int i=0;i<str.length();i++) {
            System.out.print(str.charAt(i));
        }
    }
}

class Mythread1 extends Thread {

    class_shared obj_data1;     // thread creating object for Mydata class
    Mythread1(class_shared data) {
        this.obj_data1 = data;     // connecting thread object with the shared class object
    }

    public void run() {
        obj_data1.display("Thread-1 executing      ");
    }
}

```

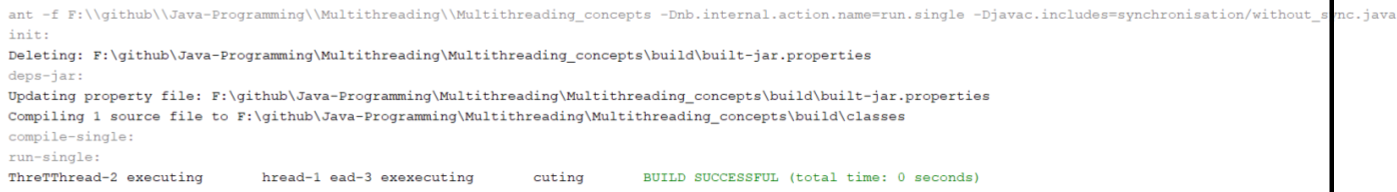


```java
class Mythread2 extends Thread {

    class_shared obj_data2;   // thread creating object for Mydata class
    Mythread2(class_shared data) {
        this.obj_data2 = data;   // connecting thread object with the shared class object
    }

    public void run() {
        obj_data2.display("Thread-2 executing      ");
    }
}

class Mythread3 extends Thread {

    class_shared obj_data3;   // thread creating object for Mydata class
    Mythread3(class_shared data) {
        this.obj_data3 = data;   // connecting thread object with the shared class object
    }

```

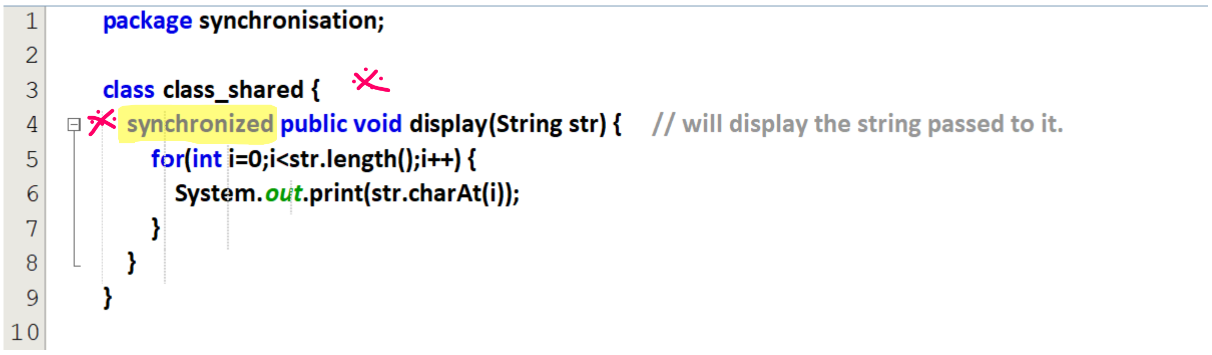


```java
    public void run() {
        obj_data3.display("Thread-3 executing      ");
    }
}

public class without_sync {
    public static void main(String args[]) {
        class_shared obj_shared = new class_shared();
        Mythread1 th1 = new Mythread1(obj_shared);
        Mythread2 th2 = new Mythread2(obj_shared);
        Mythread3 th3 = new Mythread3(obj_shared);

        th1.start();
        th2.start();
        th3.start();
    }
}
```

Output



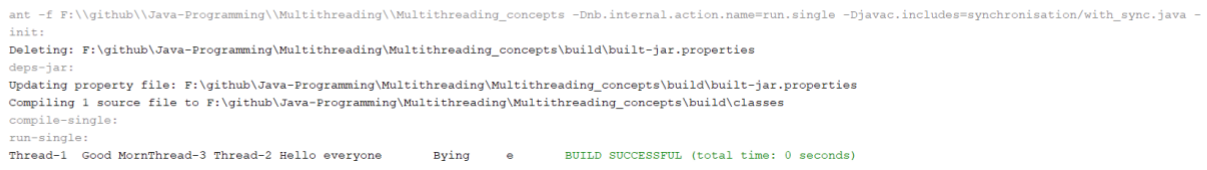
```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/without_sync.java
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
ThreTThread-2 executing       hread-1 ead-3 exexecuting       cuting       BUILD SUCCESSFUL (total time: 0 seconds)
```



```java
package synchronisation;

class class_shared {
    synchronized public void display(String str) {    // will display the string passed to it.
        for(int i=0;i<str.length();i++) {
            System.out.print(str.charAt(i));
        }
    }
}
```

Without sleep

Output

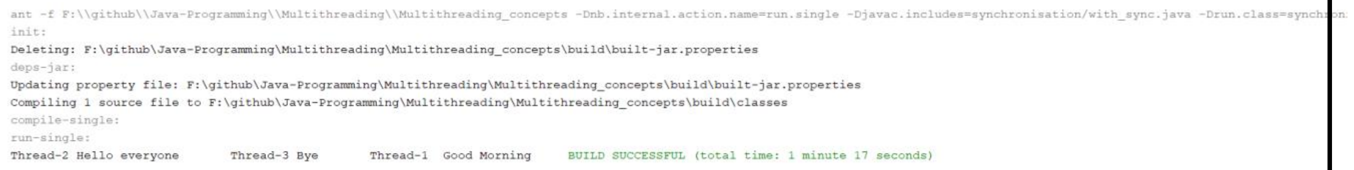


```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/with_sync.java -
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Thread-1  Good MornThread-3 Thread-2 Hello everyone       Bying     e       BUILD SUCCESSFUL (total time: 0 seconds)
```

First → thread-1, Second → thread-3, Third → thread-2

With synchronisation

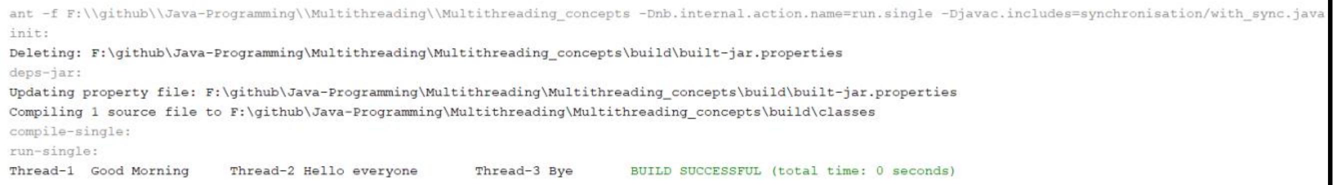
With sleep



```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/with_sync.java -Drun.class=synchron
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Thread-2 Hello everyone      Thread-3 Bye      Thread-1  Good Morning     BUILD SUCCESSFUL (total time: 1 minute 17 seconds)
```

Once a thread starts, it will continue running. It will not interrupt itself. The typing will be slow.

Without sleep

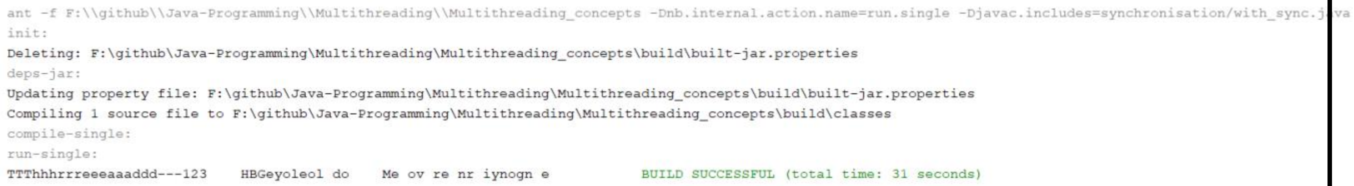


```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/with_sync.java
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Thread-1  Good Morning     Thread-2 Hello everyone      Thread-3 Bye      BUILD SUCCESSFUL (total time: 0 seconds)
```

Once a thread starts, it will continue running. It will not interrupt itself.

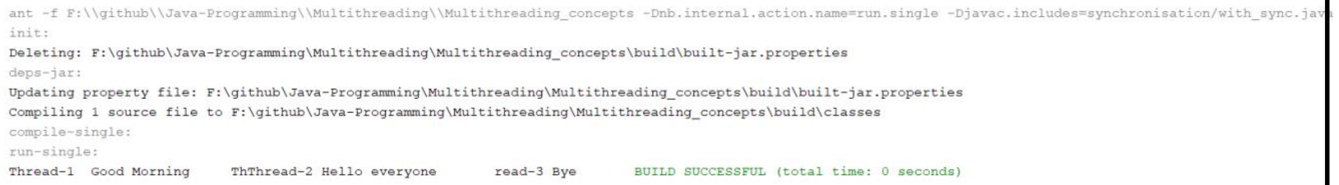
Without synchronisation

With sleep



```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/with_sync.java
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
TTThhhrrreeeaaaddd---123    HBGeyoleol do    Me ov re nr iynogn e          BUILD SUCCESSFUL (total time: 31 seconds)
```

Without sleep



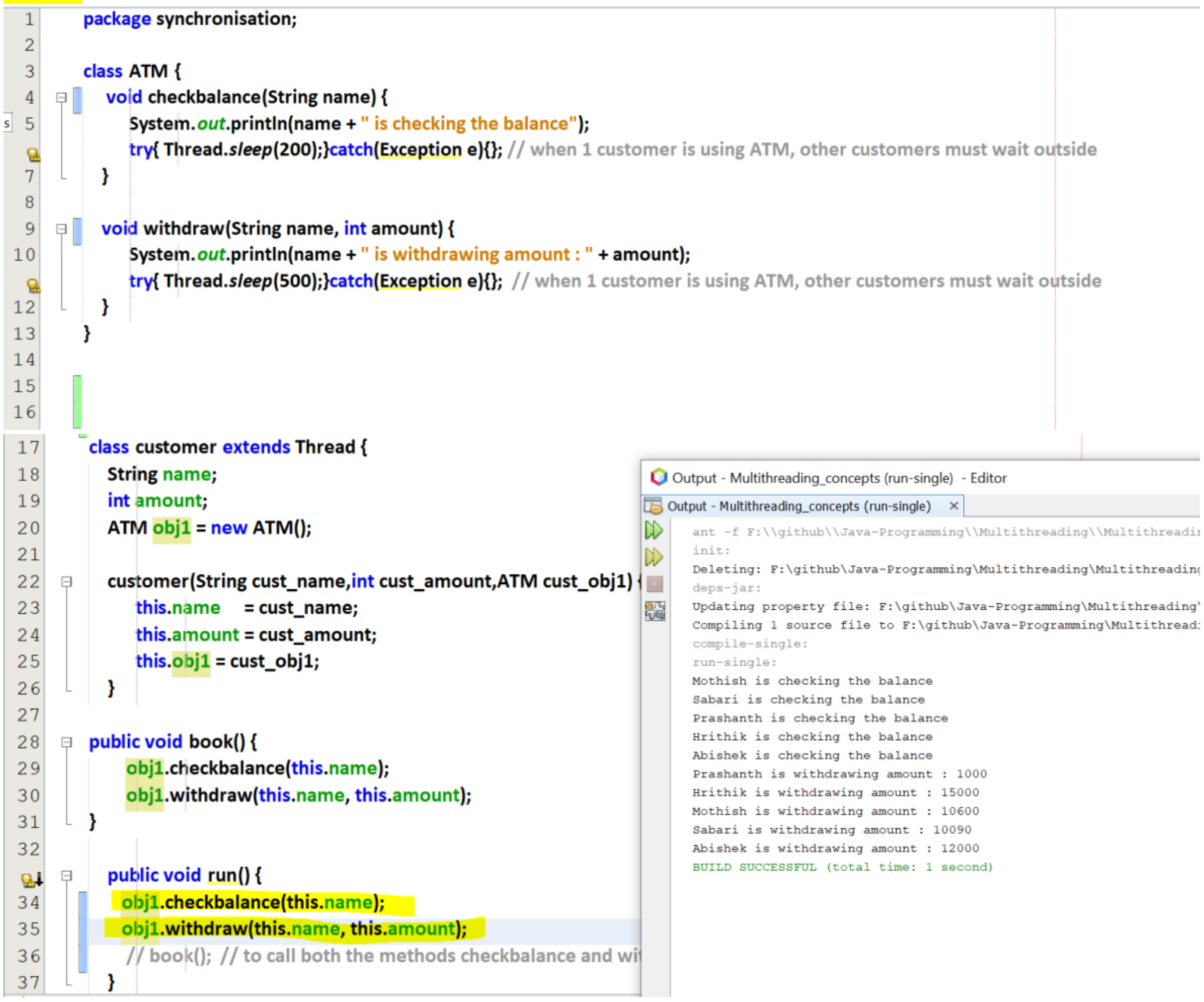
```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.includes=synchronisation/with_sync.jav
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Thread-1  Good Morning     ThThread-2 Hello everyone      read-3 Bye      BUILD SUCCESSFUL (total time: 0 seconds)
```

Here the threads starts and will not interrupt often.

Once a thread starts, it will continue to run. It will interrupt frequently because of sleep.

Students challenge ATM

Case-1



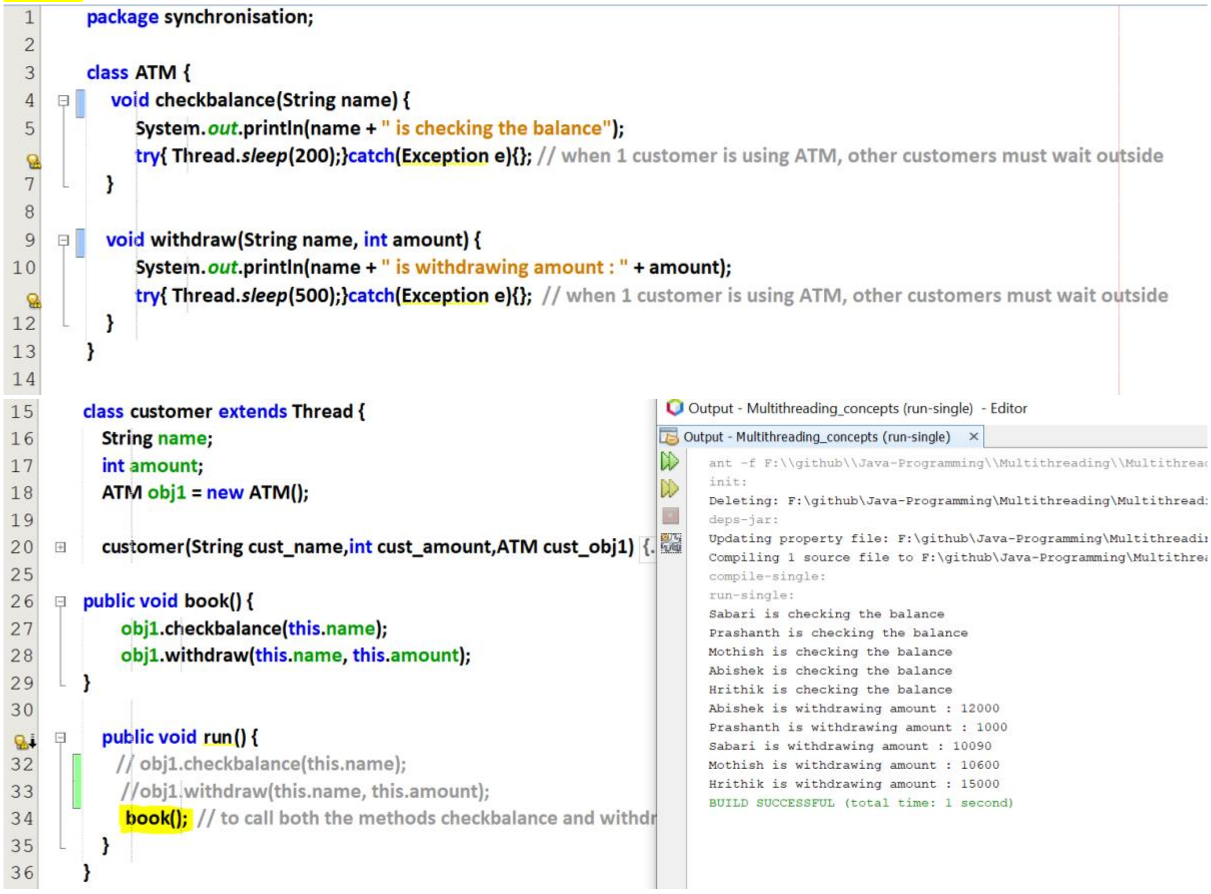
```java
package synchronisation;

class ATM {
    void checkbalance(String name) {
        System.out.println(name + " is checking the balance");
        try{ Thread.sleep(200);}catch(Exception e){}; // when 1 customer is using ATM, other customers must wait outside
    }

    void withdraw(String name, int amount) {
        System.out.println(name + " is withdrawing amount : " + amount);
        try{ Thread.sleep(500);}catch(Exception e){};  // when 1 customer is using ATM, other customers must wait outside
    }
}



class customer extends Thread {
    String name;
    int amount;
    ATM obj1 = new ATM();

    customer(String cust_name,int cust_amount,ATM cust_obj1) {
        this.name    = cust_name;
        this.amount = cust_amount;
        this.obj1 = cust_obj1;
    }

    public void book() {
        obj1.checkbalance(this.name);
        obj1.withdraw(this.name, this.amount);
    }

    public void run() {
        obj1.checkbalance(this.name);
        obj1.withdraw(this.name, this.amount);
        // book(); // to call both the methods checkbalance and wi
    }
```

Output - Multithreading_concepts (run-single) - Editor

Output - Multithreading_concepts (run-single)

```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreadi
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreadin
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\
Compiling 1 source file to F:\github\Java-Programming\Multithread:
compile-single:
run-single:
Mothish is checking the balance
Sabari is checking the balance
Prashanth is checking the balance
Hrithik is checking the balance
Abishek is checking the balance
Prashanth is withdrawing amount : 1000
Hrithik is withdrawing amount : 15000
Mothish is withdrawing amount : 10600
Sabari is withdrawing amount : 10090
Abishek is withdrawing amount : 12000
BUILD SUCCESSFUL (total time: 1 second)
```

Case-2



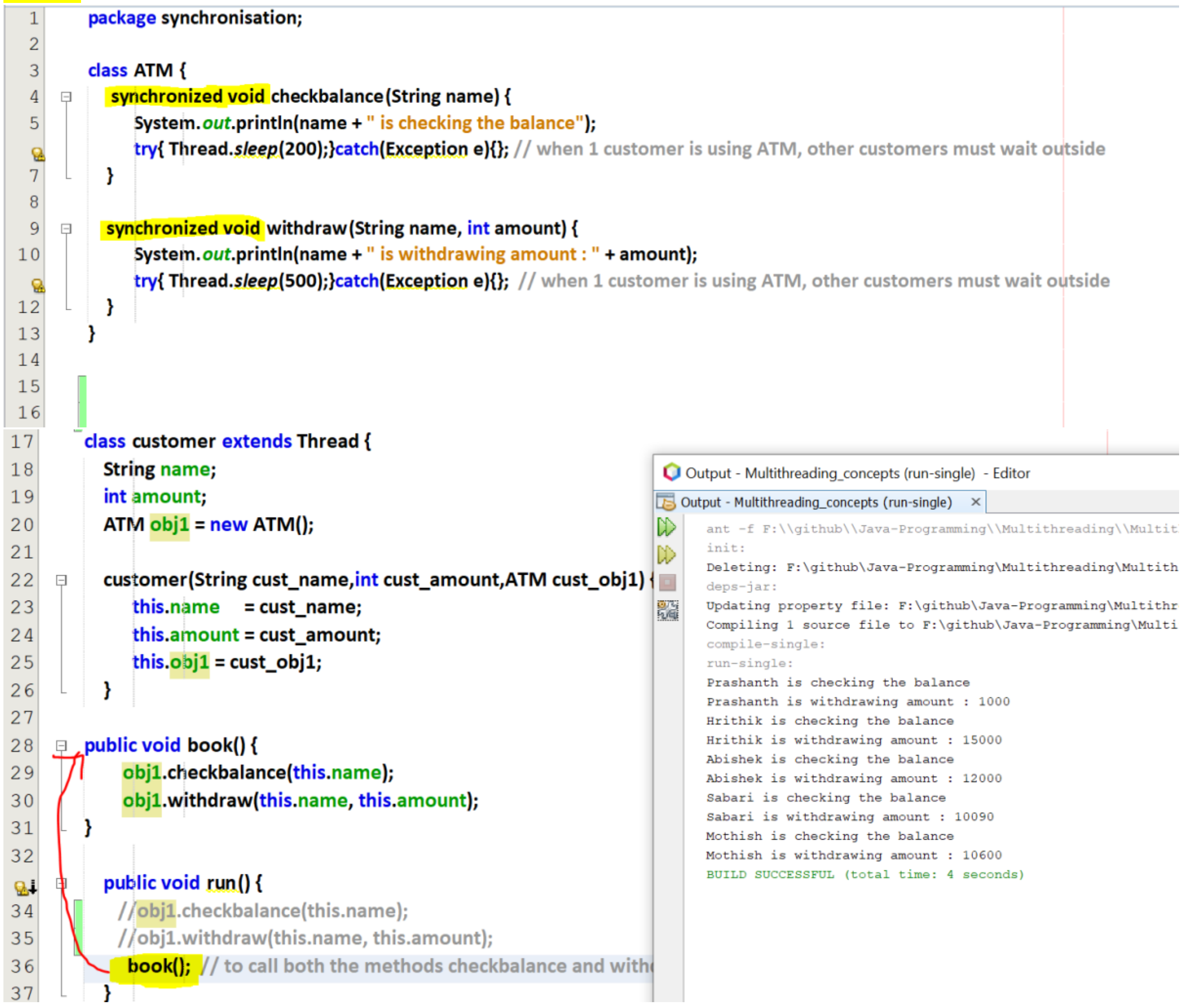
```java
package synchronisation;

class ATM {
    void checkbalance(String name) {
        System.out.println(name + " is checking the balance");
        try{ Thread.sleep(200);}catch(Exception e){}; // when 1 customer is using ATM, other customers must wait outside
    }

    void withdraw(String name, int amount) {
        System.out.println(name + " is withdrawing amount : " + amount);
        try{ Thread.sleep(500);}catch(Exception e){};  // when 1 customer is using ATM, other customers must wait outside
    }
}

class customer extends Thread {
    String name;
    int amount;
    ATM obj1 = new ATM();

    customer(String cust_name,int cust_amount,ATM cust_obj1) {.

public void book() {
        obj1.checkbalance(this.name);
        obj1.withdraw(this.name, this.amount);
}

    public void run() {
      // obj1.checkbalance(this.name);
      //obj1.withdraw(this.name, this.amount);
       book(); // to call both the methods checkbalance and withdr
    }
}
```

Output - Multithreading_concepts (run-single) - Editor

Output - Multithreading_concepts (run-single)

```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithrea
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithread
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreadi
Compiling 1 source file to F:\github\Java-Programming\Multithre
compile-single:
run-single:
Sabari is checking the balance
Prashanth is checking the balance
Mothish is checking the balance
Abishek is checking the balance
Hrithik is checking the balance
Abishek is withdrawing amount : 12000
Prashanth is withdrawing amount : 1000
Sabari is withdrawing amount : 10090
Mothish is withdrawing amount : 10600
Hrithik is withdrawing amount : 15000
BUILD SUCCESSFUL (total time: 1 second)
```

Case-3

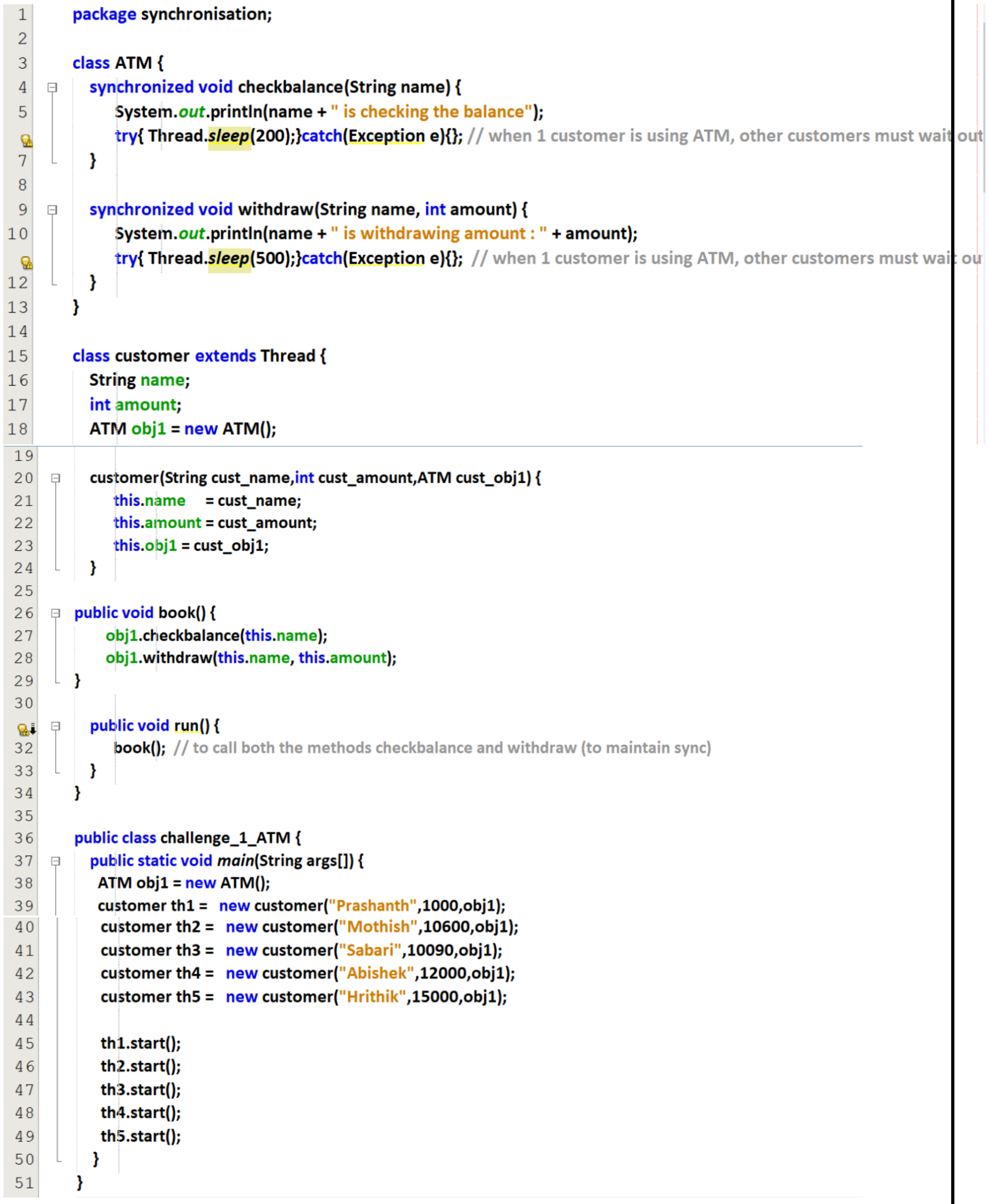


```java
package synchronisation;

class ATM {
    synchronized void checkbalance(String name) {
        System.out.println(name + " is checking the balance");
        try{ Thread.sleep(200);}catch(Exception e){}; // when 1 customer is using ATM, other customers must wait outside
    }

    synchronized void withdraw(String name, int amount) {
        System.out.println(name + " is withdrawing amount : " + amount);
        try{ Thread.sleep(500);}catch(Exception e){};  // when 1 customer is using ATM, other customers must wait outside
    }
}



class customer extends Thread {
    String name;
    int amount;
    ATM obj1 = new ATM();

    customer(String cust_name,int cust_amount,ATM cust_obj1) {
        this.name    = cust_name;
        this.amount = cust_amount;
        this.obj1 = cust_obj1;
    }

public void book() {
        obj1.checkbalance(this.name);
        obj1.withdraw(this.name, this.amount);
}

    public void run() {
      //obj1.checkbalance(this.name);
      //obj1.withdraw(this.name, this.amount);
        book();  // to call both the methods checkbalance and with
    }
```

Output - Multithreading_concepts (run-single) - Editor

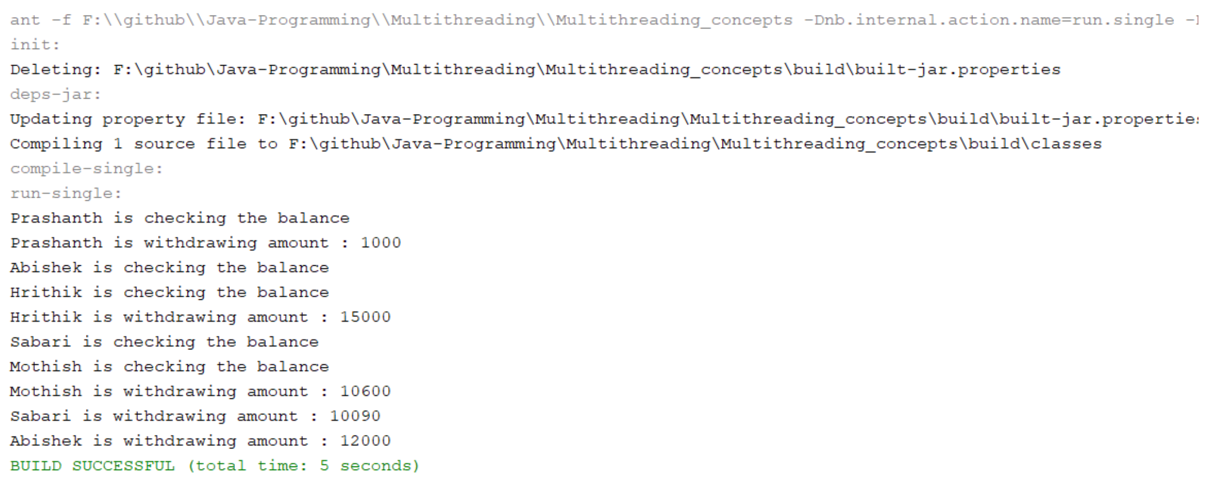
Output - Multithreading_concepts (run-single) ×

```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multit
init:
Deleting: F:\github\Java-Programming\Multithreading\Multith
deps-jar:
Updating property file: F:\github\Java-Programming\Multithr
Compiling 1 source file to F:\github\Java-Programming\Multi
compile-single:
run-single:
Prashanth is checking the balance
Prashanth is withdrawing amount : 1000
Hrithik is checking the balance
Hrithik is withdrawing amount : 15000
Abishek is checking the balance
Abishek is withdrawing amount : 12000
Sabari is checking the balance
Sabari is withdrawing amount : 10090
Mothish is checking the balance
Mothish is withdrawing amount : 10600
BUILD SUCCESSFUL (total time: 4 seconds)
```



```java
package synchronisation;

class ATM {
  synchronized void checkbalance(String name) {
      System.out.println(name + " is checking the balance");
      try{ Thread.sleep(200);}catch(Exception e){}; // when 1 customer is using ATM, other customers must wait out
  }

  synchronized void withdraw(String name, int amount) {
      System.out.println(name + " is withdrawing amount : " + amount);
      try{ Thread.sleep(500);}catch(Exception e){};  // when 1 customer is using ATM, other customers must wait ou
  }
}

class customer extends Thread {
  String name;
  int amount;
  ATM obj1 = new ATM();

  customer(String cust_name,int cust_amount,ATM cust_obj1) {
      this.name     = cust_name;
      this.amount = cust_amount;
      this.obj1 = cust_obj1;
  }

public void book() {
      obj1.checkbalance(this.name);
      obj1.withdraw(this.name, this.amount);
}

  public void run() {
      book();  // to call both the methods checkbalance and withdraw (to maintain sync)
  }
}

public class challenge_1_ATM {
  public static void main(String args[]) {
   ATM obj1 = new ATM();
   customer th1 =  new customer("Prashanth",1000,obj1);
    customer th2 =  new customer("Mothish",10600,obj1);
    customer th3 =  new customer("Sabari",10090,obj1);
    customer th4 =  new customer("Abishek",12000,obj1);
    customer th5 =  new customer("Hrithik",15000,obj1);

    th1.start();
    th2.start();
    th3.start();
    th4.start();
    th5.start();
   }
}
```

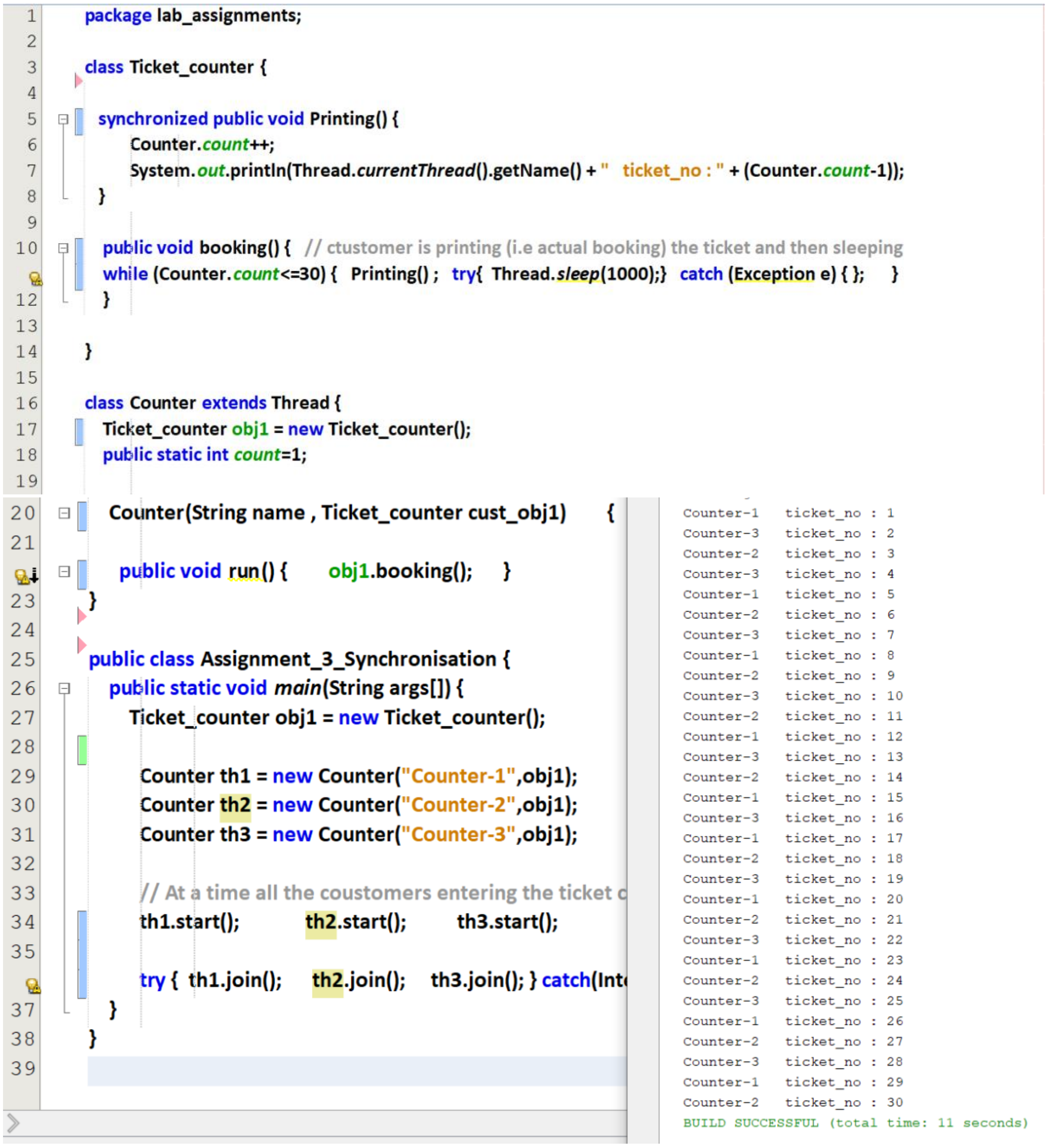
Output



```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.propertie
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Prashanth is checking the balance
Prashanth is withdrawing amount : 1000
Abishek is checking the balance
Hrithik is checking the balance
Hrithik is withdrawing amount : 15000
Sabari is checking the balance
Mothish is checking the balance
Mothish is withdrawing amount : 10600
Sabari is withdrawing amount : 10090
Abishek is withdrawing amount : 12000
BUILD SUCCESSFUL (total time: 5 seconds)
```

Students challenge Movie Ticket

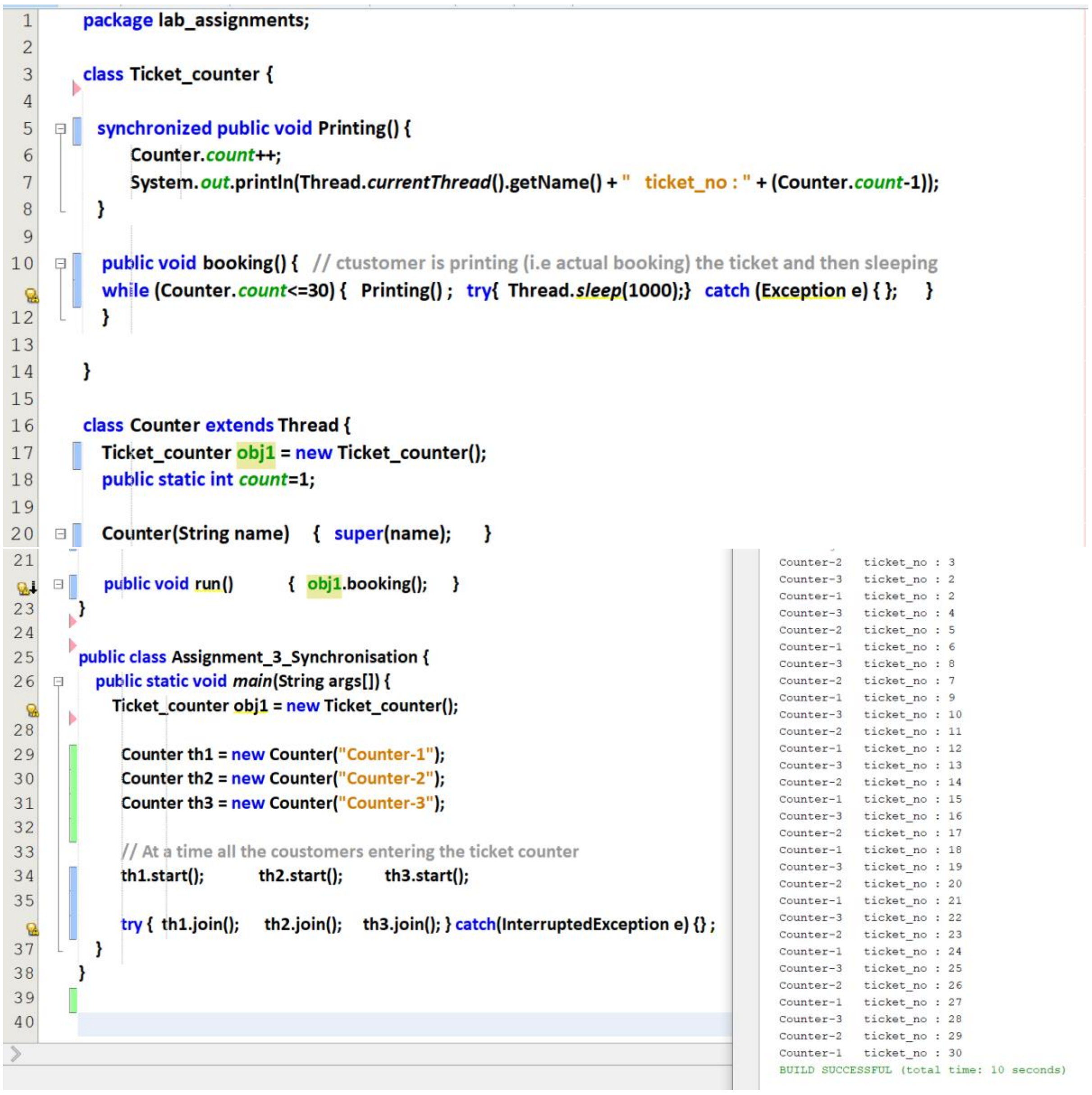
Method-1



```java
package lab_assignments;

class Ticket_counter {

  synchronized public void Printing() {
        Counter.count++;
        System.out.println(Thread.currentThread().getName() + "   ticket_no : " + (Counter.count-1));
  }

   public void booking() {  // ctustomer is printing (i.e actual booking) the ticket and then sleeping
   while (Counter.count<=30) {   Printing() ;   try{  Thread.sleep(1000);}   catch (Exception e) { };     }
   }

}

class Counter extends Thread {
   Ticket_counter obj1 = new Ticket_counter();
   public static int count=1;

   Counter(String name , Ticket_counter cust_obj1)      {

     public void run() {      obj1.booking();     }
  }

public class Assignment_3_Synchronisation {
   public static void main(String args[]) {
      Ticket_counter obj1 = new Ticket_counter();

        Counter th1 = new Counter("Counter-1",obj1);
        Counter th2 = new Counter("Counter-2",obj1);
        Counter th3 = new Counter("Counter-3",obj1);

        // At a time all the coustomers entering the ticket c
        th1.start();          th2.start();        th3.start();

        try {  th1.join();      th2.join();     th3.join(); } catch(Int
   }
 }
```

```
Counter-1   ticket_no : 1
Counter-3   ticket_no : 2
Counter-2   ticket_no : 3
Counter-3   ticket_no : 4
Counter-1   ticket_no : 5
Counter-2   ticket_no : 6
Counter-3   ticket_no : 7
Counter-1   ticket_no : 8
Counter-2   ticket_no : 9
Counter-3   ticket_no : 10
Counter-2   ticket_no : 11
Counter-1   ticket_no : 12
Counter-3   ticket_no : 13
Counter-2   ticket_no : 14
Counter-1   ticket_no : 15
Counter-3   ticket_no : 16
Counter-1   ticket_no : 17
Counter-2   ticket_no : 18
Counter-3   ticket_no : 19
Counter-1   ticket_no : 20
Counter-2   ticket_no : 21
Counter-3   ticket_no : 22
Counter-1   ticket_no : 23
Counter-2   ticket_no : 24
Counter-3   ticket_no : 25
Counter-1   ticket_no : 26
Counter-2   ticket_no : 27
Counter-3   ticket_no : 28
Counter-1   ticket_no : 29
Counter-2   ticket_no : 30
BUILD SUCCESSFUL (total time: 11 seconds)
```

Method-2



```java
package lab_assignments;

class Ticket_counter {

 synchronized public void Printing() {
      Counter.count++;
      System.out.println(Thread.currentThread().getName() + "   ticket_no : " + (Counter.count-1));
 }

  public void booking() {  // ctustomer is printing (i.e actual booking) the ticket and then sleeping
  while (Counter.count<=30) {  Printing() ;  try{  Thread.sleep(1000);}  catch (Exception e) { };     }
  }

}

class Counter extends Thread {
  Ticket_counter obj1 = new Ticket_counter();
  public static int count=1;

  Counter(String name)   {  super(name);     }

   public void run()        {  obj1.booking();    }
}

public class Assignment_3_Synchronisation {
   public static void main(String args[]) {
     Ticket_counter obj1 = new Ticket_counter();

      Counter th1 = new Counter("Counter-1");
      Counter th2 = new Counter("Counter-2");
      Counter th3 = new Counter("Counter-3");

      // At a time all the coustomers entering the ticket counter
      th1.start();         th2.start();       th3.start();

      try {  th1.join();     th2.join();     th3.join(); } catch(InterruptedException e) {} ;
   }
}
```

```
Counter-2   ticket_no : 3
Counter-3   ticket_no : 2
Counter-1   ticket_no : 2
Counter-3   ticket_no : 4
Counter-2   ticket_no : 5
Counter-1   ticket_no : 6
Counter-3   ticket_no : 8
Counter-2   ticket_no : 7
Counter-1   ticket_no : 9
Counter-3   ticket_no : 10
Counter-2   ticket_no : 11
Counter-1   ticket_no : 12
Counter-3   ticket_no : 13
Counter-2   ticket_no : 14
Counter-1   ticket_no : 15
Counter-3   ticket_no : 16
Counter-2   ticket_no : 17
Counter-1   ticket_no : 18
Counter-3   ticket_no : 19
Counter-2   ticket_no : 20
Counter-1   ticket_no : 21
Counter-3   ticket_no : 22
Counter-2   ticket_no : 23
Counter-1   ticket_no : 24
Counter-3   ticket_no : 25
Counter-2   ticket_no : 26
Counter-1   ticket_no : 27
Counter-3   ticket_no : 28
Counter-2   ticket_no : 29
Counter-1   ticket_no : 30
BUILD SUCCESSFUL (total time: 10 seconds)
```

Race Condition

Inter-thread communication

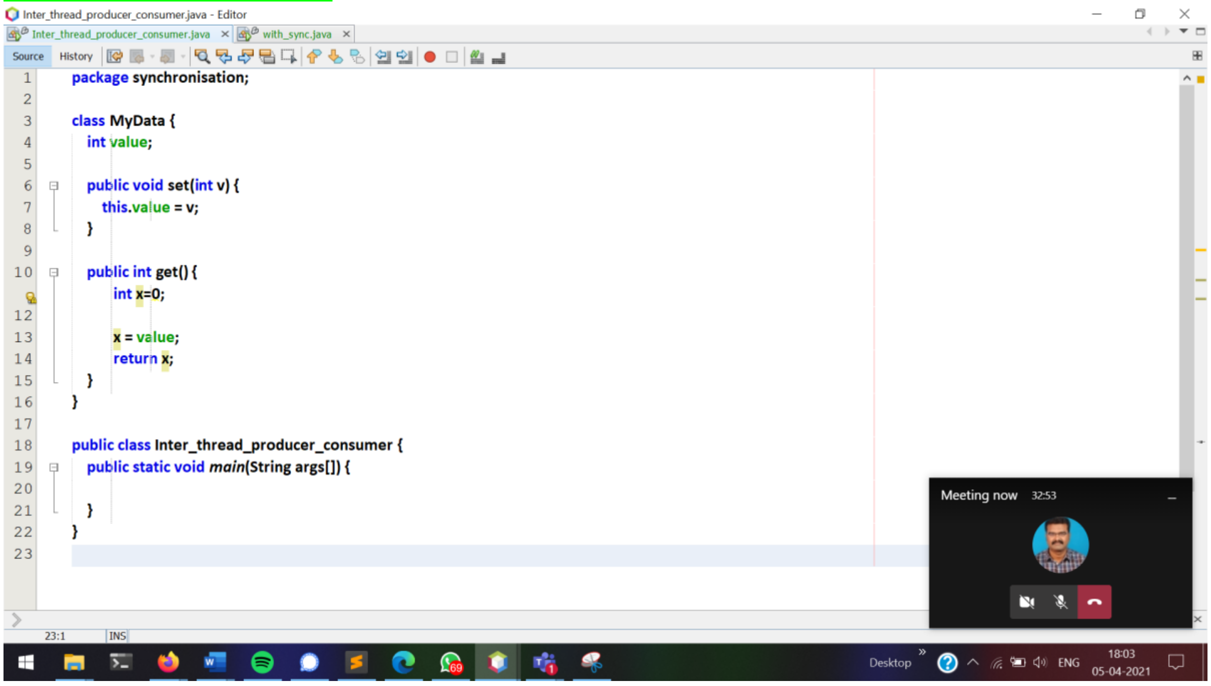
Similar to inter-process communication in OS, here in JAVA it is inter-thread communication..
Before this JAVA achieved synchronisation, Now the programmer has to achieve inter-thread communication.

producer will write the value using set().
consumer will read the value using get().

flag = true (Producer’s turn)
flag = false (Consumer’s turn)

notify() → will wake up only one blocked thread
notify_all() → will wake up all the blocked threads

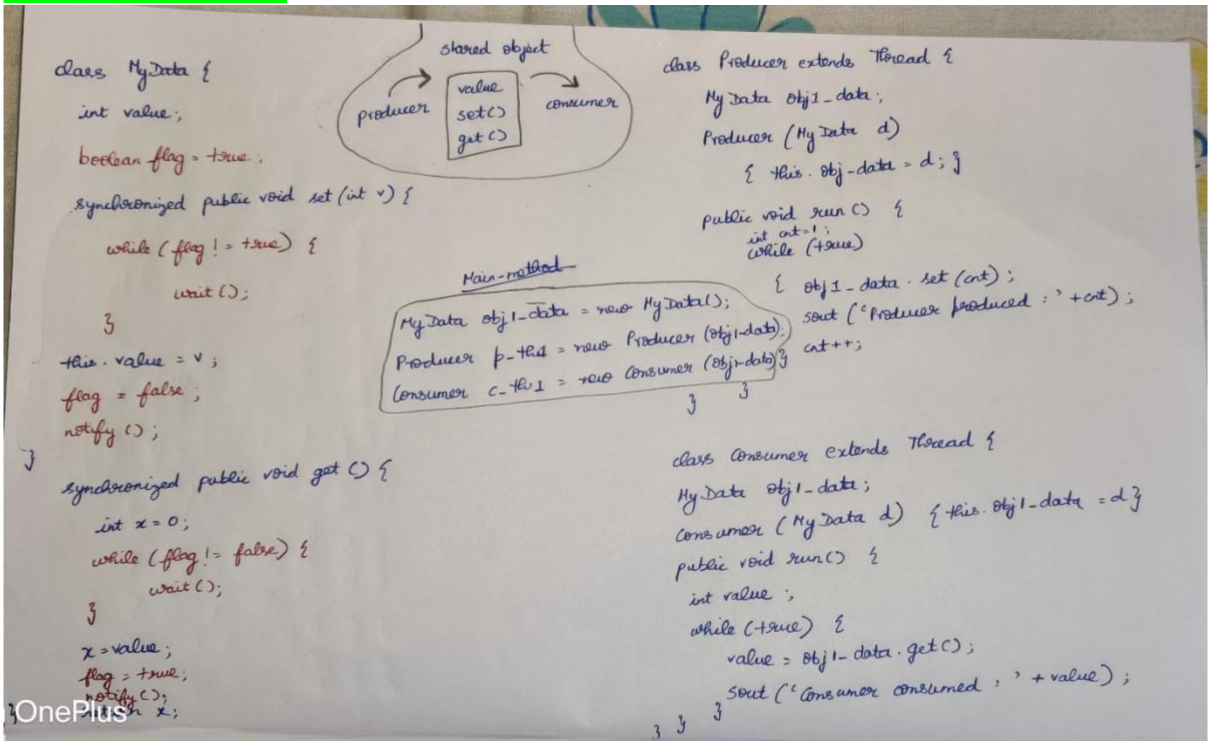
Without synchronisation


Inter_thread_producer_consumer.java - Editor
Inter_thread_producer_consumer.java
with_sync.java
Source
History
Meeting now 32:53
23:1
INS
Desktop
ENG
18:03
05-04-2021

```java
package synchronisation;

class MyData {
    int value;

    public void set(int v) {
        this.value = v;
    }

    public int get() {
        int x=0;

        x = value;
        return x;
    }
}

public class Inter_thread_producer_consumer {
    public static void main(String args[]) {

    }
}

```

Producer produces items, but consumer consumes already consumed items.

With synchronisation


shared object
producer
value
set()
get()
consumer
Main-method

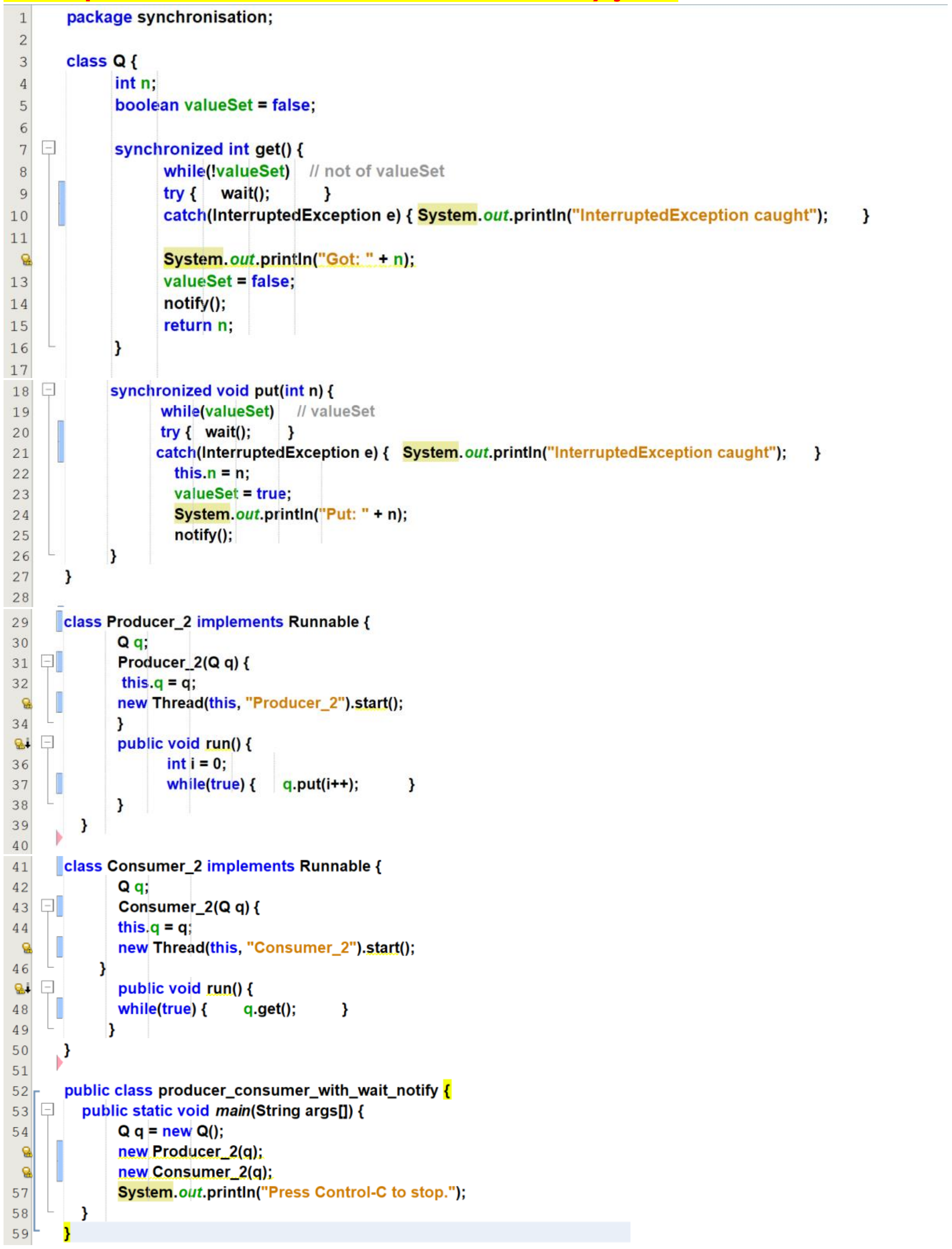
```java
class MyData {
  int value;
  boolean flag = true;
  synchronized public void set (int v) {
      while (flag != true) {
          wait ();
      }
    this.value = v;
    flag = false;
    notify ();
}
  synchronized public void get () {
     int x = 0;
     while (flag != false) {
          wait ();
     }
     x = value;
     flag = true;
     notify ();
     return x;
}
```

```java
MyData obj1_data = new MyData();
Producer p_th1 = new Producer (obj1_data);
Consumer c_th1 = new Consumer (obj1_data)}
```

```java
class Producer extends Thread {
    MyData obj1_data;
    Producer (MyData d)
        { this.obj_data = d; }

    public void run () {
        int cnt=1;
        while (true)
          { obj1_data.set (cnt);
            sout ("Producer produced : " +cnt);
            cnt++;
    }
  }
```

```java
class Consumer extends Thread {
  MyData obj1_data;
  Consumer (MyData d) {this.obj1_data = d}
  public void run() {
   int value ;
   while (true) {
      value = obj1_data.get();
      sout ("Consumer consumed : " + value);
   }
} }
```

Refer producer_consumer_with_wait_notify.java

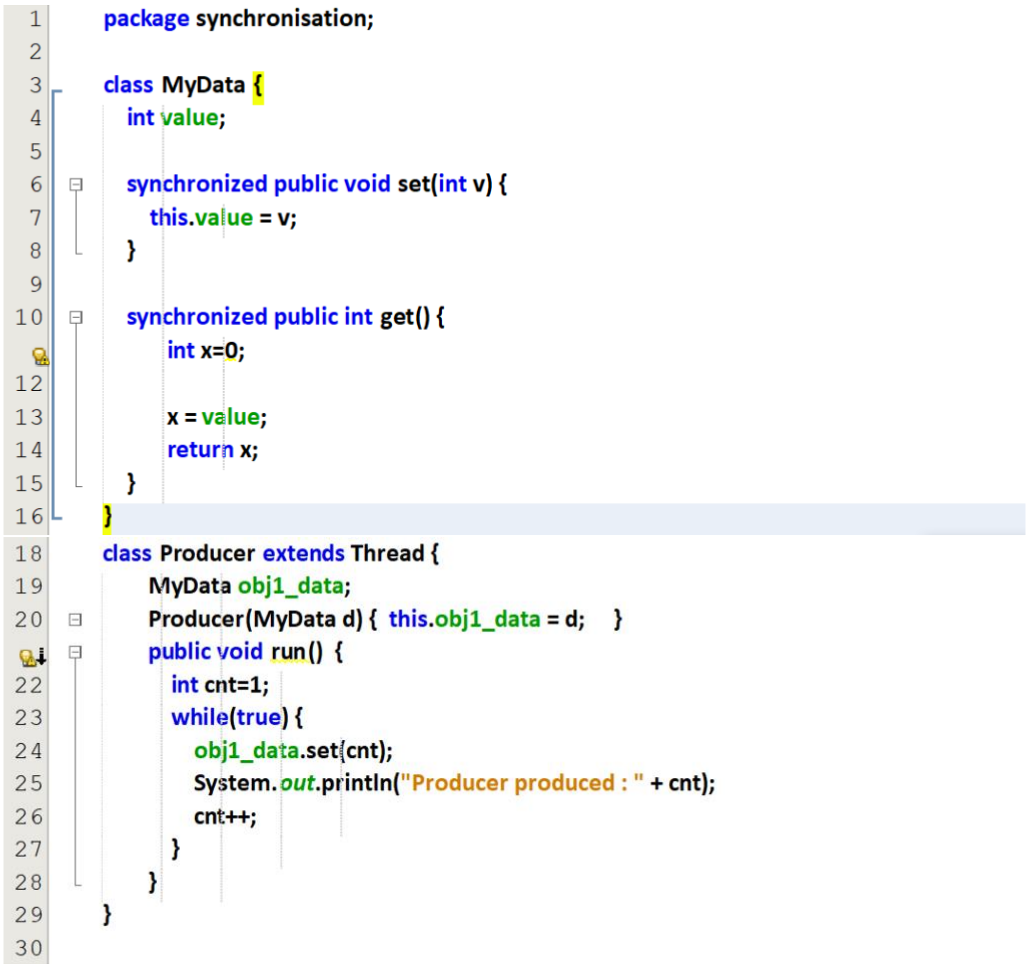


```java
package synchronisation;

class Q {
        int n;
        boolean valueSet = false;

        synchronized int get() {
                while(!valueSet)    // not of valueSet
                try {     wait();          }
                catch(InterruptedException e) { System.out.println("InterruptedException caught");      }

                System.out.println("Got: " + n);
                valueSet = false;
                notify();
                return n;
        }

        synchronized void put(int n) {
                while(valueSet)     // valueSet
                try {   wait();       }
                catch(InterruptedException e) {   System.out.println("InterruptedException caught");     }
                  this.n = n;
                  valueSet = true;
                  System.out.println("Put: " + n);
                  notify();
        }
}

class Producer_2 implements Runnable {
        Q q;
        Producer_2(Q q) {
         this.q = q;
        new Thread(this, "Producer_2").start();
        }
        public void run() {
                int i = 0;
                while(true) {     q.put(i++);         }
        }
   }

class Consumer_2 implements Runnable {
        Q q;
        Consumer_2(Q q) {
        this.q = q;
        new Thread(this, "Consumer_2").start();
     }
        public void run() {
        while(true) {       q.get();         }
      }
}

public class producer_consumer_with_wait_notify {
   public static void main(String args[]) {
        Q q = new Q();
        new Producer_2(q);
        new Consumer_2(q);
        System.out.println("Press Control-C to stop.");
   }
}
```

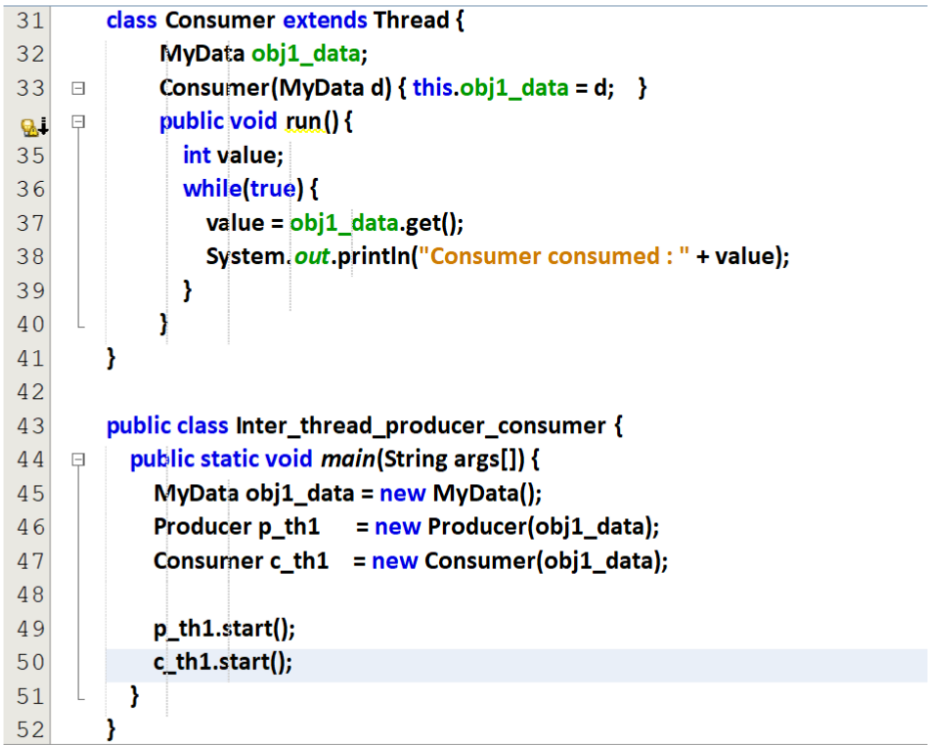
Output:

```
Press Control-C to stop.
Put: 0
Got: 0
Put: 1
Got: 1
Put: 2
Got: 2
Put: 3
Got: 3
Put: 4
Got: 4
Put: 5
```

Refer producer_consumer.java

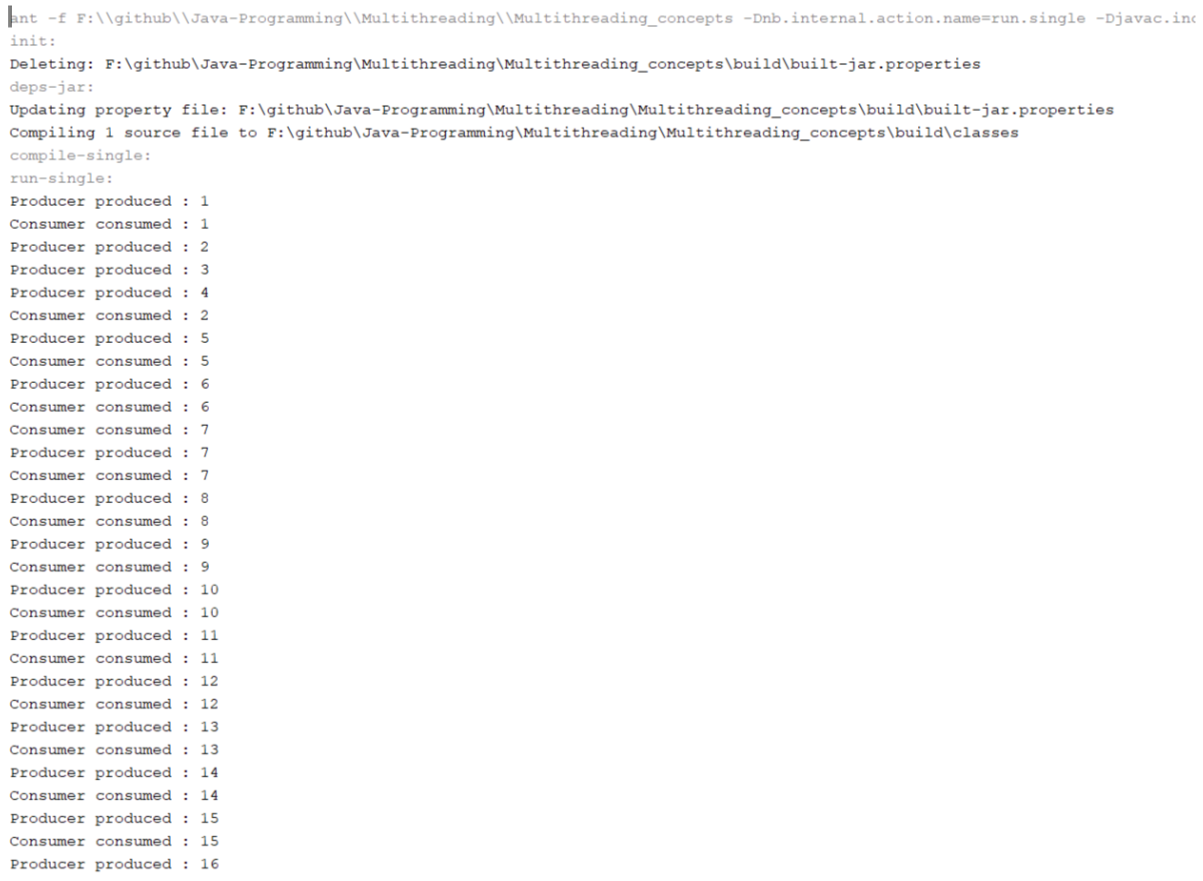


```java
package synchronisation;

class MyData {
   int value;

   synchronized public void set(int v) {
      this.value = v;
   }

   synchronized public int get() {
        int x=0;

        x = value;
        return x;
   }
}
class Producer extends Thread {
      MyData obj1_data;
      Producer(MyData d) {  this.obj1_data = d;    }
      public void run()  {
         int cnt=1;
         while(true) {
            obj1_data.set(cnt);
            System.out.println("Producer produced : " + cnt);
            cnt++;
         }
      }
}

```



```java
class Consumer extends Thread {
       MyData obj1_data;
       Consumer(MyData d) { this.obj1_data = d;    }
       public void run() {
          int value;
          while(true) {
             value = obj1_data.get();
             System.out.println("Consumer consumed : " + value);
          }
       }
}

public class Inter_thread_producer_consumer {
   public static void main(String args[]) {
      MyData obj1_data = new MyData();
      Producer p_th1      = new Producer(obj1_data);
      Consumer c_th1    = new Consumer(obj1_data);

      p_th1.start();
      c_th1.start();
   }
}
```

Output



```
ant -f F:\\github\\Java-Programming\\Multithreading\\Multithreading_concepts -Dnb.internal.action.name=run.single -Djavac.inc
init:
Deleting: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
deps-jar:
Updating property file: F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\built-jar.properties
Compiling 1 source file to F:\github\Java-Programming\Multithreading\Multithreading_concepts\build\classes
compile-single:
run-single:
Producer produced : 1
Consumer consumed : 1
Producer produced : 2
Producer produced : 3
Producer produced : 4
Consumer consumed : 2
Producer produced : 5
Consumer consumed : 5
Producer produced : 6
Consumer consumed : 6
Consumer consumed : 7
Producer produced : 7
Consumer consumed : 7
Producer produced : 8
Consumer consumed : 8
Producer produced : 9
Consumer consumed : 9
Producer produced : 10
Consumer consumed : 10
Producer produced : 11
Consumer consumed : 11
Producer produced : 12
Consumer consumed : 12
Producer produced : 13
Consumer consumed : 13
Producer produced : 14
Consumer consumed : 14
Producer produced : 15
Consumer consumed : 15
Producer produced : 16
```

Even-though synchronisation is there for both set() and get(), these are 2 independent methods, so we cannot find the sync.

Students challenge Teacher, student and WhiteBoard

Teacher writes statement-1 in the board.
4 Students reads that statement-1 (i.e reading from the board and writing into their note book)

After read by 4 students then only teacher will write the statement-2. Until the student reads, the teacher should wait.

When teacher writes end, students will vacate the class and the program completes.

WhiteBoard class will not allow the teacher to write until all the students read.