

Threads

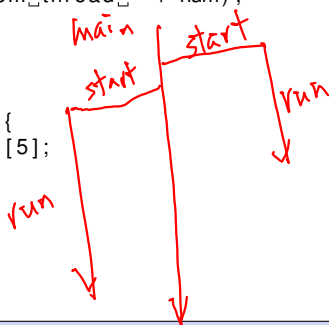
Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, June 8-9, 2010

A Basic Example

```
1  class HelloThread extends Thread {
2      int num;
3      HelloThread(int n){ num = n; }
4      public void run() {
5          for(int t = 0; t < 10; t++){
6              System.out.println(t + ":_Hello_from_thread_" + num);
7          }
8      }
9
10 public class ThreadDemo1{
11     public static void main(String [] argv){
12         HelloThread[] tarr = new HelloThread[5];
13         for(int i=0;i<5;i++){
14             tarr[i] = new HelloThread(i);
15             tarr[i].start();
16         }
17     }
18 }
```



Another Route: Runnable

```
1  class HelloRunnable implements Runnable {
2      int num;
3      HelloRunnable(int n){ num = n; }
4      public void run() {
5          for(int t = 0; t < 10; t++)
6              System.out.println(t + ":_Hello_from_thread_" + num);
7      }
8  }
9
10 public class ThreadDemo2{
11     public static void main(String[] argv){
12         Thread[] tarr = new Thread[5];
13
14         for(int i=0;i<5;i++){
15             HelloRunnable r = new HelloRunnable(i);
16             tarr[i] = new Thread(r);
17             tarr[i].start();
18         }
19     }
20 }
```

Slow Things Down: `sleep`

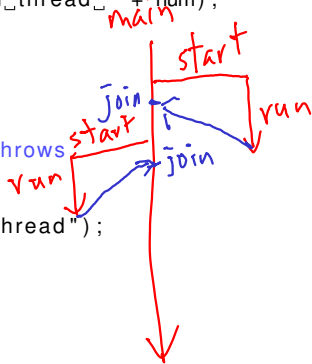
```
1 public class ThreadDemo3{
2     public static void main(String[] argv)
3         throws InterruptedException{
4
5         for(int i=0;i<10;i++){
6             System.out.println(i);
7             Thread.sleep(1000);
8         }
9     }
10 }
```

Wake Up: interrupt

```
1  class HelloThread extends Thread {
2      int num; Thread tr;
3      HelloThread(int n, Thread _tr){ num = n; tr = _tr; }
4      public void run() {
5          for(int t = 0; t < 10; t++){
6              System.out.println(t + ":_Hello_from_thread_" + num);
7              tr.interrupt();
8              try{ Thread.sleep(500); }
9              catch(InterruptedException e){
10                 System.out.println("the_sleep_of_" + num + "_interrupted");
11             }
12         }
13     }
14 }
15 public class ThreadDemo4{
16     public static void main(String[] ar) throws InterruptedException{
17         HelloThread[] tarr = new HelloThread[5];
18         for(int i=0;i<5;i++){
19             tarr[i] = new HelloThread(i,
20                 (i == 0? Thread.currentThread() : tarr[i-1]));
21             tarr[i].start();
22         }
23     }
24 }
```

Wait for Finish: join

```
1 class HelloThread extends Thread {
2     int num;
3     HelloThread(int n){ num = n; }
4     public void run() {
5         for(int t = 0; t < 10; t++){
6             System.out.println(t + " : _Hello_ from _thread_" + num);
7         }
8     }
9 }
10
11 public class ThreadDemo5{
12     public static void main(String[] argv) throws
13         InterruptedException{
14         for(int i=0;i<3;i++){
15             System.out.println("Starting_a_new_thread");
16             HelloThread t = new HelloThread(i);
17             t.start();
18             t.join();
19         }
20     }
```



Story of a Bank: Part I

Once upon a time, a bank uses the following system to allow customers to spend in local stores easily

```
1     localcredit = getCredit(customer);
2     tospend = getPrice(item);
3     if (tospend <= localcredit){
4         newcredit = localcredit - tospend;
5         notifyNewCredit(newcredit);
6     }
```

Story of a Bank: Part II

Normally,

```
1   localcredit1 = getCredit(customer1); //10000
2   tospend1 = getPrice(item1); //3000
3   if (tospend1 <= localcredit1){
4       newcredit1 = localcredit1 - tospend1; //2000 7000
5       notifyNewCredit(newcredit1);
6   }
7   localcredit2 = getCredit(customer2); //10000
8   tospend2 = getPrice(item2); //2000
9   if (tospend2 <= localcredit2){
10      newcredit2 = localcredit2 - tospend2; //8000
11      notifyNewCredit(newcredit2);
12  }
```


Story of a Bank: Part III

Unfortunately, customer 1 and 2 share the same account but go to different stores

```
1   localcredit1 = getCredit(customer1); //10000
2   localcredit2 = getCredit(customer2); //10000
3   tospend1 = getPrice(item1); //3000
4   if (tospend1 <= localcredit1){
5       newcredit1 = localcredit1 - tospend1; //7000
6       notifyNewCredit(newcredit1);
7   }
8   tospend2 = getPrice(item2); //2000
9   if (tospend2 <= localcredit2){
10      newcredit2 = localcredit2 - tospend2; //8000
11      notifyNewCredit(newcredit2);
12  }
13  getCredit(customer1); //8000
14  getCredit(customer2); //8000 8000
```

Local copies are not trustworthy. Must update global copy **atomically**

An Example with Counter Threads I

```
1 class Counter{
2     private int c = 0;
3     private int ic, dc;
4     private void sleep(){
5         try{ Thread.sleep(200); }
6         catch(Exception e){ System.err.println(e); }
7     }
8
9     public void inc(){ ic++; sleep();
10        int newc = c + 1; sleep(); c = newc;}
11    public void dec(){ dc++; sleep();
12        int newc = c - 1; sleep(); c = newc;}
13    public void info(){
14        System.out.println(ic + "__" + dc + "_=" + c);
15    }
16 }
17
18
19
20
21
```

`ic++; //1`
`dc++; //1`
`newc1 = c + 1; //1`
`newc2 = c - 1; //-1`
`c = newc1; //1`
`c = newc2; //-1`

An Example with Counter Threads II

```
22 class IncCounterThread extends Thread{
23     Counter c;
24     IncCounterThread(Counter c){ this.c = c;}
25     public void run(){
26         while(true){
27             //synchronized(c){
28                 {
29                     c.inc();
30                     c.info();
31                 }
32             try{
33                 Thread.sleep(100);
34             }
35             catch(Exception e){
36             }
37         }
38     }
39 }
40
41
42
43
```

```
synchronized(c){
synchronized(d){
synchronized(e){
...
}
}
}
```

An Example with Counter Threads III

```
44 class DecCounterThread extends Thread{
45     Counter c;
46     DecCounterThread(Counter c){ this.c = c;}
47     public void run(){
48         while(true){
49             //synchronized(c){
50                 {
51                     c.dec();
52                     c.info();
53                 }
54                 try{
55                     Thread.sleep(100);
56                 }
57                 catch(Exception e){
58                 }
59             }
60         }
61     }
62
63
64
65
```

An Example with Counter Threads IV

```
66 public class CounterDemo{
67     public static void main(String [] argv){
68         Counter c = new Counter();
69         IncCounterThread plus = new IncCounterThread(c);
70         DecCounterThread minus = new DecCounterThread(c);
71
72         plus.start();
73         minus.start();
74     }
75 }
```

Story of a Couple: Part I

Once upon a time, a couple shares a credit card account. To prevent overdraft, they agreed on the following protocol for using the credit card:

```
1   tospend = getPrice(item);  
2   currentlimit = checkCreditbyCellphone();  
3   if (tospend <= currentlimit)  
4       do_transaction(); // atomically
```

Story of a Couple: Part II

Normally,

```
1   tospend = getPrice(item); //by George: 50000
2   currentlimit = checkCreditbyCellphone(); //60000
3   if (tospend <= currentlimit) //by Mary: yes
4       do_transaction(); //atomically
5   tospend = getPrice(item); //by Mary: 20000
6   currentlimit = checkCreditbyCellphone(); //10000
7   if (tospend <= currentlimit) //by Mary: no
8       do_transaction(); //atomically
```


Story of a Couple: Part III

Unfortunately,

```
1  tospend = getPrice(item); //by George: 50000
2  currentlimit = checkCreditbyCellphone(); //60000
3  //George drives to the store
4  tospend = getPrice(item); //by Mary: 20000
5  currentlimit = checkCreditbyCellphone(); //60000
6  if (tospend <= currentlimit) //by Mary: yes
7     do_transaction();
8  if (tospend <= currentlimit) //by George: yes
9     do_transaction(); //OVERDRAFT!!
```

Spent should happen **immediately** after checking

An Example with Couple Threads I

```
1  class NegativeException extends Exception{
2      NegativeException(double value){
3          super("Negative_value_" + value + "_not_allowed.");
4      }
5  }
6
7  class CreditCard{
8      int credit = 60000;
9
10     public int getcredit(){ return credit; }
11     public synchronized void spend(int amount) throws
12         NegativeException{
13         int newcredit = credit - amount;
14         credit = newcredit;
15
16         if (credit < 0)
17             throw new NegativeException(credit);
18     }
19 }
20 }
```

synchronized(this)-

An Example with Couple Threads II

```
21 class Person extends Thread{
22     int tospend;
23     CreditCard c;
24     Person(int tospend, CreditCard c){ this.tospend = tospend;
        this.c = c;}
25     void spend() throws NegativeException, InterruptedException{
26         synchronized(c){
27             int credit = c.getcredit();
28             Thread.sleep(100);
29             if (credit >= tospend){
30                 c.spend(tospend);
31             }
32         }
33     }
34     synchronized void spend_wrong() throws NegativeException,
        InterruptedException{
35         int credit = c.getcredit();
36         Thread.sleep(100);
37         if (credit >= tospend){
38             c.spend(tospend);
39         }
40     }
```

An Example with Couple Threads III

```
41
42     void spend_wrong_equiv() throws NegativeException ,
43         InterruptedException {
44         synchronized(this){
45             int credit = c.getcredit();
46             Thread.sleep(100);
47             if (credit >= tospend){
48                 c.spend(tospend);
49             }
50     }
51
52     public void run(){
53         try{
54             spend();
55         }
56         catch(Exception e){
57             System.out.println(e);
58         }
59     }
60 }
61
```

An Example with Couple Threads IV

```
62 public class CreditCardDemo{
63     public static void main(String [] argv){
64         for(int i = 0; i < Integer.parseInt(argv[0]); i++){
65             CreditCard c = new CreditCard();
66             Person George = new Person(50000, c);
67             Person Mary = new Person(20000, c);
68
69             George.start();
70             Mary.start();
71         }
72     }
73 }
```

Synchronization

`synchronized`: binds operations altogether (with respect to a lock)

- synchronized method: the lock is the class (for static method) or the object (for non-static method)
 - usually used to protect the variables within the class/object
- synchronized block: the lock is explicitly provided
 - flexible, fine-grained use

```
syn(c){  
    syn(d){  
    }  
}
```

More on the Lock

- after getting the lock, can “use” any synchronized method/block that depends on the lock
- lock releases after the method/block finishes (by return or exception)

A Story of the Black and White Goats: Deadlock



```
synchronized(b){  
    synchronized(w){  
        gogogogogogo();  
    }  
}
```

```
synchronized(w){  
    synchronized(b){  
        gogogogogogo();  
    }  
}
```