Threads

Hsuan-Tien Lin

Department of CSIE, NTU

OOP Class, May 16, 2013

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  }
```

```
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  }
```

```
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    }
```

```
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
           InterruptedException{
13      for(int i=0;i<3;i++){
14        System.out.println("Starting_a_new_thread");
15        HelloThread t = new HelloThread(i);
16        t.start();
17        t.join();
18      }
19    }
20  }
```

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          }
```

Normally,

```
1      localcredit1 = getCredit(customer1); //10000
2      tospend1 = getPrice(item1); //3000
3      if (tospend1 <= localcredit1){
4        newcredit1 = localcredit1 - tospend1; //2000
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); //7000
```

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 synchronized void inc(){
10              ic++; sleep();
11              int newc = c + 1; sleep(); c = newc;
12      }
13      public synchronized void dec(){
14          dc++; sleep();
15          int newc = c - 1; sleep(); c = newc;
16      }
17      public synchronized void info(){
18          System.out.println(ic + "_-_" + dc + "_=_" + c);
19      }
20  }
21
```

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

```
44              while ( true ) {
45                  {
46                      c . dec ( ) ;
47                      c . info ( ) ;
48                  }
49                  try {
50                      Thread . sleep (100) ;
51                  }
52                  catch ( Exception e ) {
53                  }
54              }
55          }
56  }
57
58  public class CounterDemo {
59      public static void main ( String [ ] argv ) {
60          Counter c = new Counter ( ) ;
61          IncCounterThread plus = new IncCounterThread ( c ) ;
62          DecCounterThread minus = new DecCounterThread ( c ) ;
63
64          plus . start ( ) ;
65          minus . start ( ) ;
```

```
66          }
67      }
```

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
```

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
```

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

```
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
            NegativeException{
12          int newcredit = credit - amount;
13          credit = newcredit;
14
15          if (credit < 0)
16              throw new NegativeException(credit);
17      }
18  }
19
20
```

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      }
```

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

```
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  }
```

`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

- 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)

Ways to be Polite

```
1        synchronized void make_payment(int amount){
2          while(no_money()){
3            this.complain(1000);
4          }
5        }
```

```
1        synchronized void make_payment(int amount){
2          while(no_money()){
3            Thread.sleep(1000);
4          }
5        }
6        // ...
```

```
1        synchronized void make_payment(int amount){
2          while(no_money()){
3            this.wait(1000);
4          }
5        }
6        // ...
```

```
1        synchronized void make_payment(int amount){
2          while(no_money()){
3            this.wait();
4          }
5        }
6
7        synchronized void get_money(int amount){
8          money += amount;
9          notifyAll();
10       }
```

- difference between `wait` and `sleep`: the former releases the lock temporarily