Java

面向对象程序设计

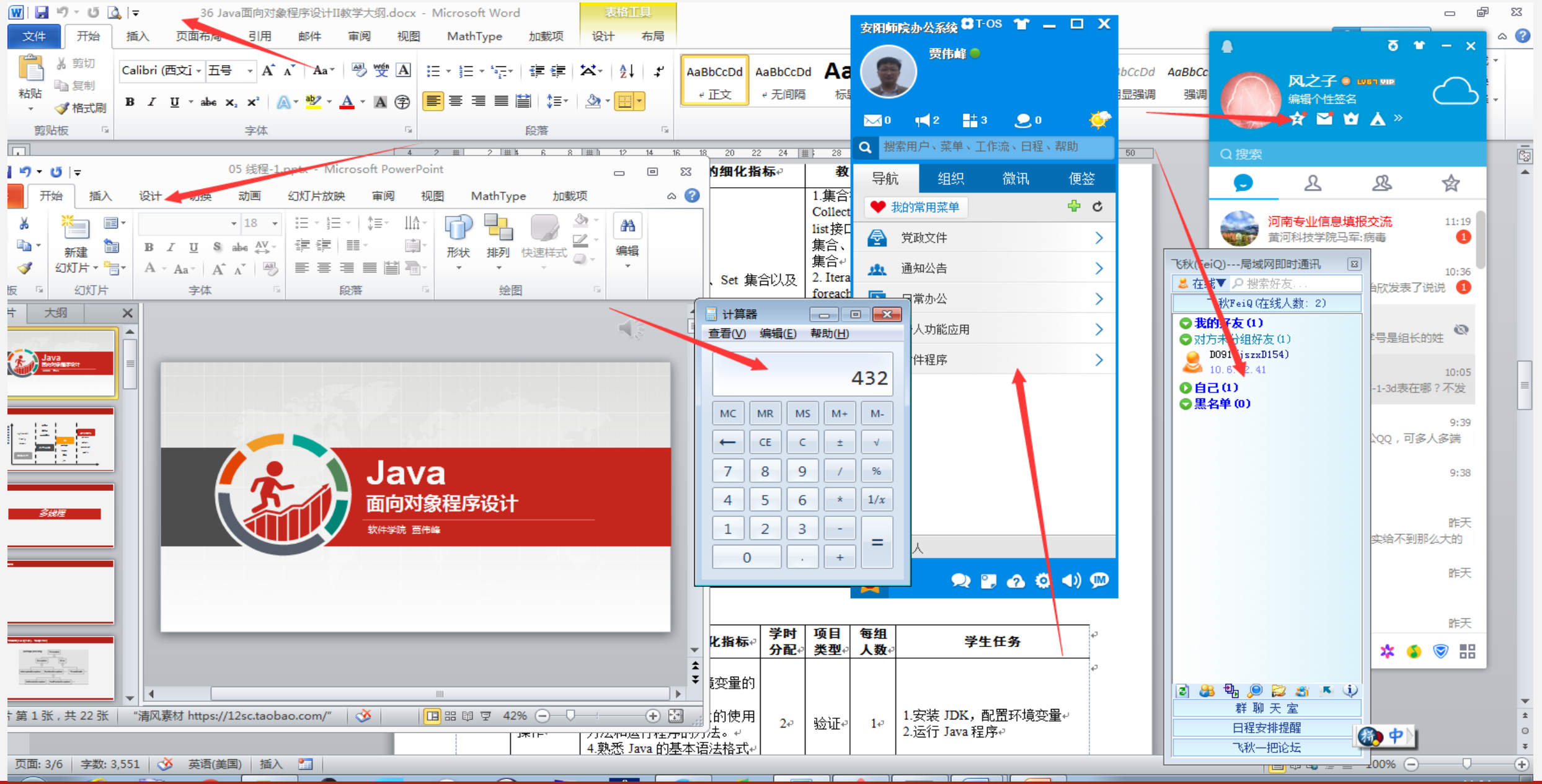
软件学院 贾伟峰

复习

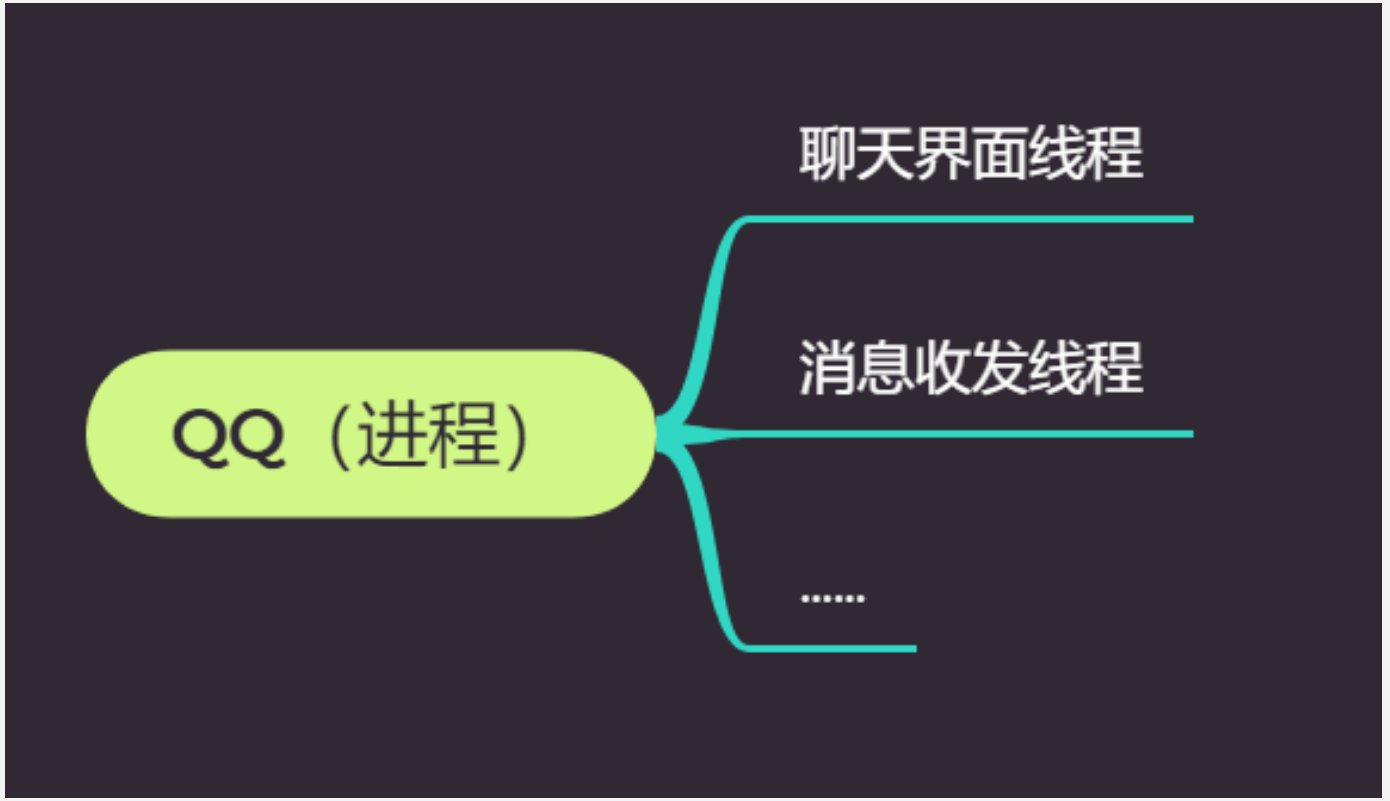

第四次课的内容
try{}catch{}
finally
throws
捕捉异常
编译时
运行时
自定义异常
异常分类
包
package
import
用途
jar
访问控制
private
default
protected
public

多线程

进程（程序运行起来，就是进程；计算机同时运行多个程序，即多个进程并发执行。）



线程（进一步讲，线程是进程内部可以并发执行的调度单位）



进程“重”、线程“轻”，都有“并发”执行的特点。

我们什么时候用线程?

Java中的非多线程编程（普通编程、单线程）

```
1  public class Example01
2  {
3      public static void main(String[] args){
4          MyThread myThread = new MyThread();
5          myThread.run();
6          while(true){
7              System.out.println("Main方法在执行");
8          }
9      }
10 }
11 class MyThread
12 {
13     public void run(){
14         while(true){
15             System.out.println("MyThread类的run()方法在运行。");
16         }
17     }
18 }
```

永远得不到执行，why?

Java中的多线程编程（并发）

```
1  public class Example02
2  {
3      public static void main(String[] args){
4          MyThread myThread = new MyThread();
5          myThread.start();
6          while(true){
7              System.out.println("main()方法在运行");
8          }
9      }
10 }
11 class MyThread extends Thread
12 {
13     public void run(){
14         while(true){
15             System.out.println("MyThread类的run方法在运行");
16         }
17     }
18 }
```

```
MyThread类的run方法在运行
main()方法在运行
main()方法在运行
MyThread类的run方法在运行
main()方法在运行
MyThread类的run方法在运行
main()方法在运行
MyThread类的run方法在运行
main()方法在运行
main()方法在运行
MyThread类的run方法在运行
main()方法在运行
MyThread类的run方法在运行
main()方法在运行
main()方法在运行
main()方法在运行
main()方法在运行
MyThread类的run方法在运行
```

线程就是继承自Thread的子类，重写run方法，实例化为对象，然后调用start方法运行。相当于进程在此“兵分两路”。

可是Java是单继承啊，这意味着继承后的子类无法作为线程执行啊.....

Thread(Runnable target)

另外一种方法实现线程，不再受单继承的限制！

```
1  public class Example03
2  {
3      public static void main(String[] args){
4          MyThread myThread = new MyThread();
5          Thread thread = new Thread(myThread);
6          thread.start();
7          while(true){
8              System.out.println("main 方法在运行");
9          }
10     }
11 }
12
13 class MyThread implements Runnable
14 {
15     public void run(){
16         while(true){
17             System.out.println("MyThread run方法运行");
18         }
19     }
20 }
```

多了一步

写一个多线程程序，模拟4个窗口卖票。

多线程模拟4个窗口卖票

```
1  public class Example04
2  {
3      public static void main(String[] args){
4          new TicketWindow().start();
5          new TicketWindow().start();
6          new TicketWindow().start();
7          new TicketWindow().start();
8      }
9  }
10 class TicketWindow extends Thread
11 {
12     private int tickets = 100;
13     public void run(){
14         while(true){
15             if(tickets > 0){
16                 Thread th = Thread.currentThread();
17                 String th_name = th.getName();
18                 System.out.println(th_name + "正在发售第 " + tickets -- +"张票");
19             }
20         }
21     }
22 }
```

分别启动4个线程

只能是单继承

余票数

当前线程

```
Thread-0正在发售第 55张票
Thread-0正在发售第 54张票
Thread-0正在发售第 53张票
Thread-0正在发售第 52张票
Thread-1正在发售第 67张票
Thread-2正在发售第 79张票
Thread-2正在发售第 78张票
Thread-1正在发售第 66张票
Thread-1正在发售第 65张票
Thread-1正在发售第 64张票
Thread-0正在发售第 51张票
Thread-0正在发售第 50张票
Thread-0正在发售第 49张票
Thread-0正在发售第 48张票
Thread-3正在发售第 86张票
Thread-3正在发售第 85张票
```

多线程模拟4个窗口卖票

```java
public class Example05
{
    public static void main(String[] args){
        TicketWindow tw = new TicketWindow();
        new Thread(tw, "窗口1").start();
        new Thread(tw, "窗口2").start();
        new Thread(tw, "窗口3").start();
        new Thread(tw, "窗口4").start();
    }
}
class TicketWindow implements Runnable
{
    private int tickets = 100;
    public void run(){
        while(true){
            if(tickets > 0){
                Thread th = Thread.currentThread();
                String th_name = th.getName();
                System.out.println(th_name + "正在发售" + tickets-- +"张票");
            }
        }
    }
}
```

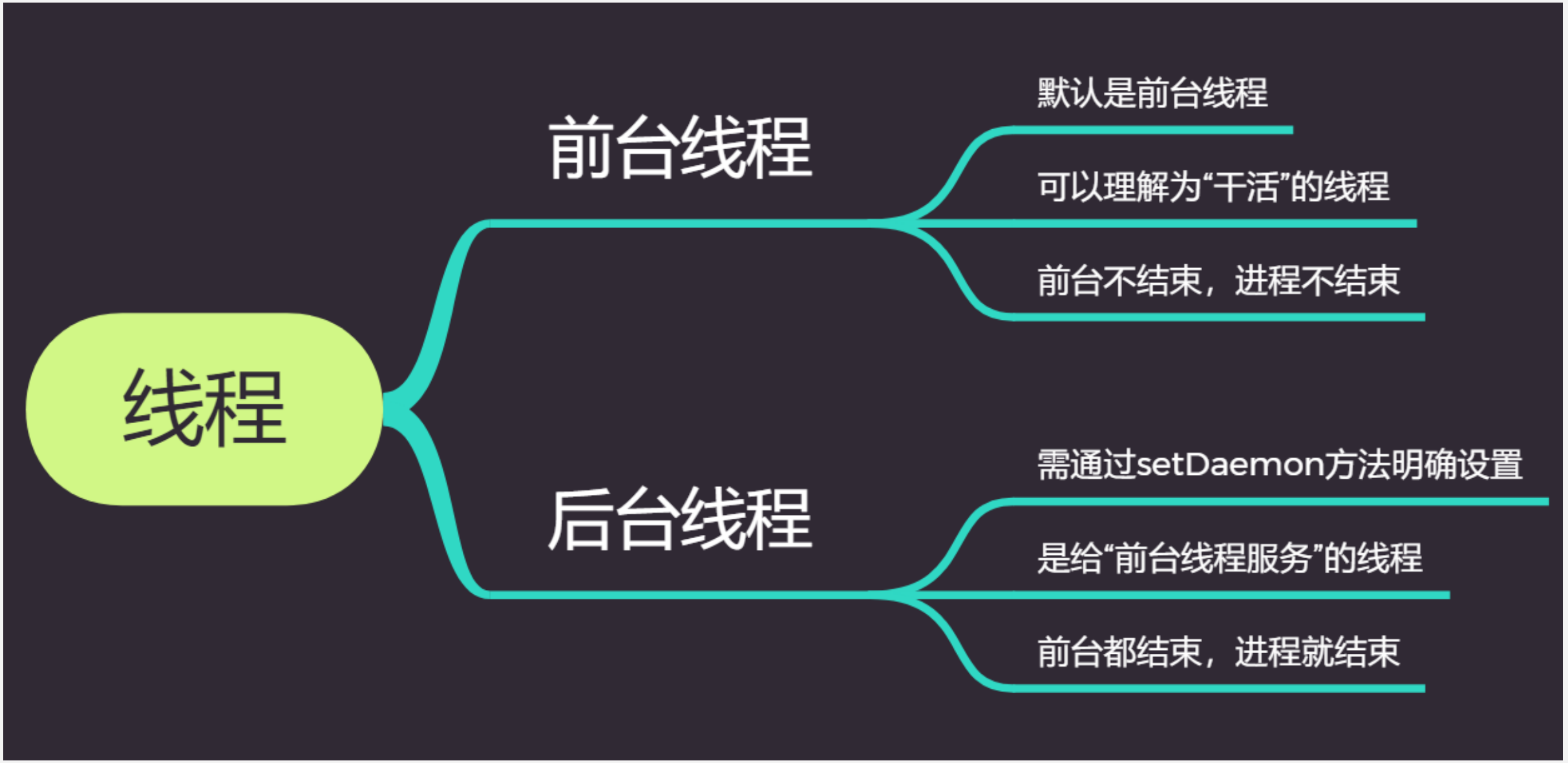
借助tw同时开启了4个线程，
共享余额tickets。

```
窗口1正在发售100张票
窗口1正在发售98张票
窗口1正在发售95张票
窗口1正在发售94张票
窗口2正在发售99张票
窗口1正在发售93张票
窗口4正在发售96张票
窗口3正在发售97张票
窗口4正在发售90张票
窗口1正在发售91张票
窗口2正在发售92张票
窗口1正在发售87张票
窗口4正在发售88张票
窗口3正在发售89张票
窗口4正在发售84张票
窗口1正在发售85张票
窗口2正在发售86张票
```

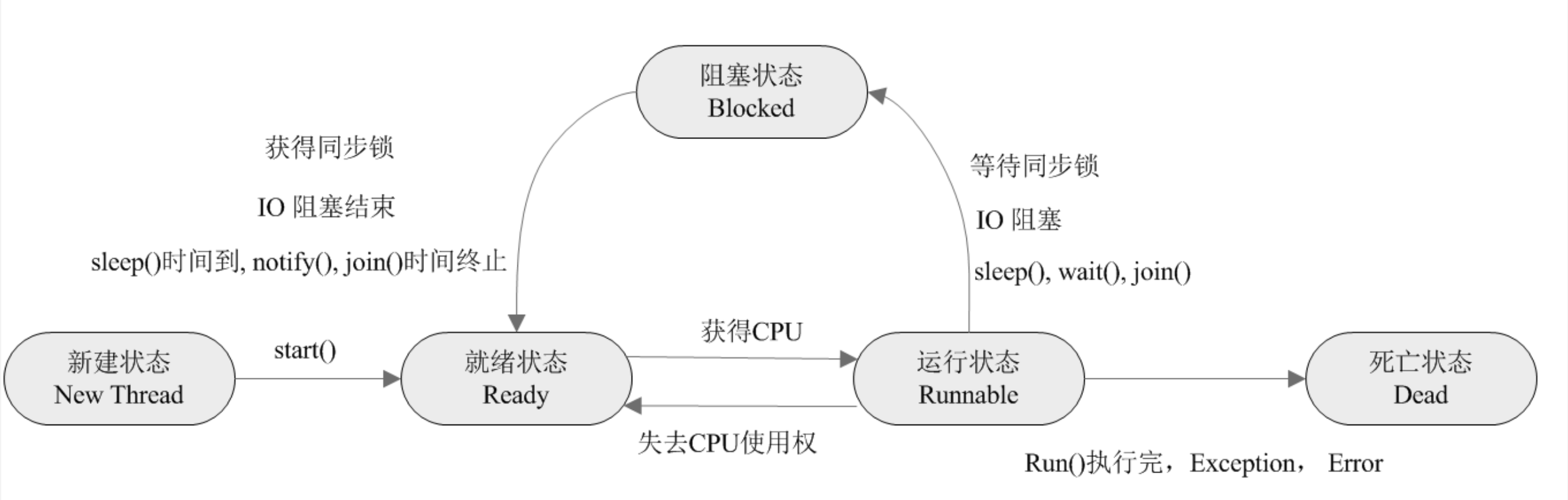
or

卖的票数一样吗?

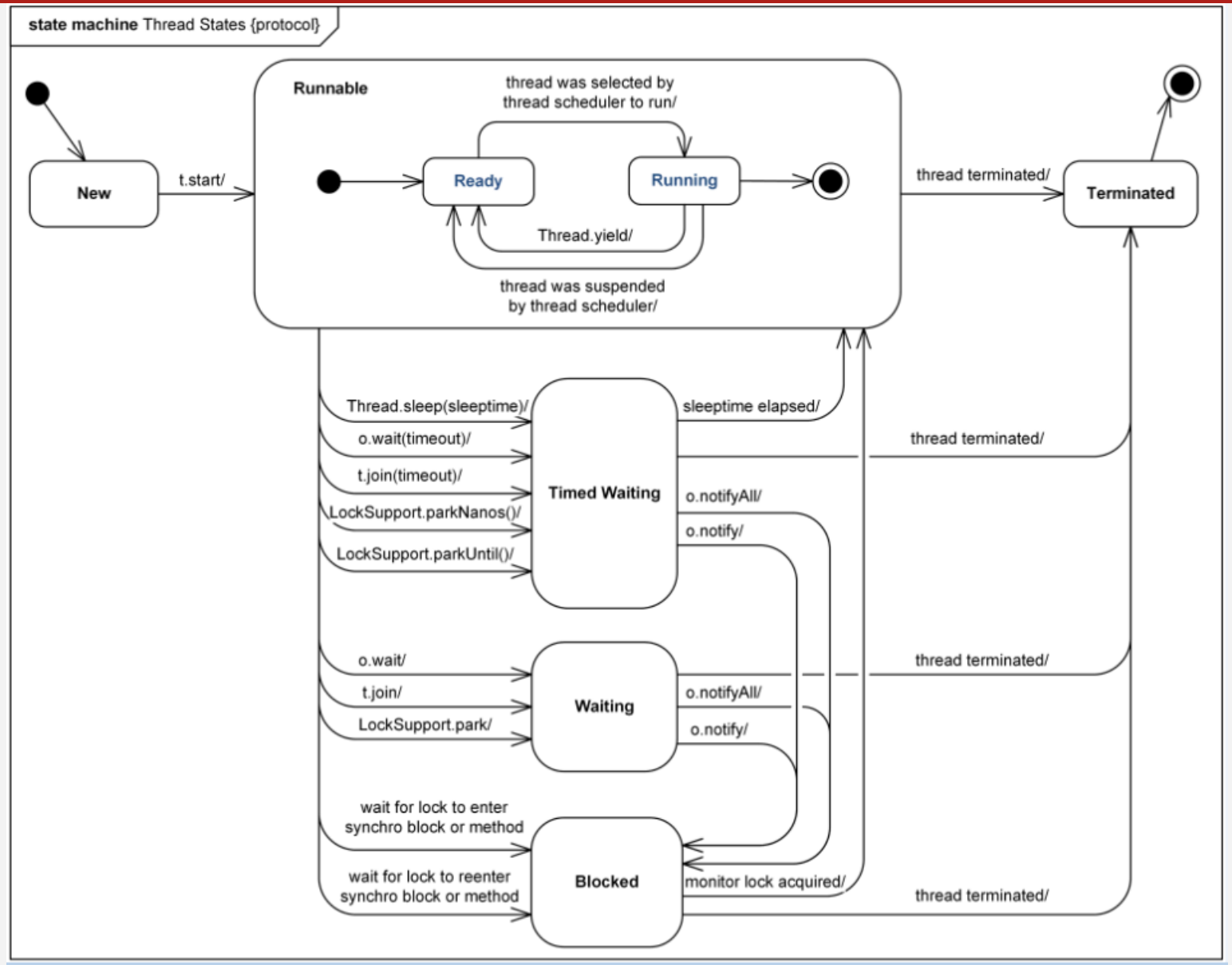
前台线程与后台线程（了解）p172



线程生命周期及状态转换



线程生命周期及状态转换（英文详细版）



线程多了如何调度？谁先谁后？能不能让步？能不能插队？

谁先谁后（优先级）？

线程的调度——基于优先级的调度

```java
public class Example07
{
    public static void main(String[] args)
    {
        Thread minPriority = new Thread(new MinPriority(), "优先级较低的线程");
        Thread maxPriority = new Thread(new MaxPriority(), "优先级较高的线程");
        minPriority.setPriority(Thread.MIN_PRIORITY);
        maxPriority.setPriority(10);

        maxPriority.start();
        minPriority.start();
    }
}

class MaxPriority implements Runnable
{
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println(Thread.currentThread().getName() + "正在输出： " +i);
        }
    }
}
class MinPriority implements Runnable
{
    public void run(){
        for(int i = 0; i < 10; i++){
            System.out.println(Thread.currentThread().getName() + "正在输出： " +i);
        }
    }
}
```

```
优先级较高的线程正在输出： 0
优先级较高的线程正在输出： 1
优先级较高的线程正在输出： 2
优先级较高的线程正在输出： 3
优先级较低的线程正在输出： 0
优先级较高的线程正在输出： 4
优先级较高的线程正在输出： 5
优先级较低的线程正在输出： 1
优先级较高的线程正在输出： 6
优先级较高的线程正在输出： 7
优先级较低的线程正在输出： 2
优先级较高的线程正在输出： 8
优先级较高的线程正在输出： 9
优先级较低的线程正在输出： 3
优先级较低的线程正在输出： 4
优先级较低的线程正在输出： 5
优先级较低的线程正在输出： 6
优先级较低的线程正在输出： 7
优先级较低的线程正在输出： 8
优先级较低的线程正在输出： 9
```

主动休眠（真睡，会“阻塞”），让出资源。

线程的调度——线程休眠

```java
public class Example08
{
    public static void main(String[] args){
        new Thread(new SleepThread()).start();
        for(int i = 1; i <=10; i++){
            if(i == 5){
                try{
                    Thread.sleep(2000);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }

            System.out.println("主线程正在输出" + i);

            try{
                    Thread.sleep(500);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
        }

    }
}
```

```java
class SleepThread implements Runnable
{
    public void run(){
        for(int i = 1; i <=10; i++){
            if(i == 3){
                try{
                    Thread.sleep(2000);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
            }

            System.out.println("线程一正在输出：" + i);

            try{
                    Thread.sleep(500);
                }catch(InterruptedException e){
                    e.printStackTrace();
                }
        }


    }
}
```

```
主线程正在输出1
线程一正在输出：1
线程一正在输出：2
主线程正在输出2
主线程正在输出3
主线程正在输出4
线程一正在输出：3
线程一正在输出：4
线程一正在输出：5
主线程正在输出5
线程一正在输出：6
主线程正在输出6
线程一正在输出：7
主线程正在输出7
线程一正在输出：8
主线程正在输出8
线程一正在输出：9
主线程正在输出9
线程一正在输出：10
主线程正在输出10
```

主动让出（不阻塞，重新参与调度）。

线程的调度——线程让步

```
1  class YieldThread extends Thread
2  {
3      public YieldThread(String name){
4          super(name);
5      }
6      public void run(){
7          for(int i = 0; i < 5; i++){
8              System.out.println(Thread.currentThread().getName() + "---" + i);
9              if(i == 3){
10                 System.out.print("线程让步");
11                 Thread.yield();
12             }
13         }
14     }
15 }
16
17 public class Example09
18 {
19     public static void main(String[] args){
20         Thread t1 = new YieldThread("线程A");
21         Thread t2 = new YieldThread("线程B");
22         t1.start();
23         t2.start();
24     }
25 }
```

类似Thread.sleep()方法,但yield不会阻塞当前线程,而是让其转为就绪状态继续参与调度.

```
线程A---0
线程A---1
线程B---0
线程B---1
线程B---2
线程B---3
线程让步线程A---2
线程B---4
线程A---3
线程让步线程A---4
```

线程的世界里，是可以“插队”的

线程的调度——线程插队

```
1  public class Example10
2  {
3      public static void main(String[] args){
4          Thread t = new Thread(new EmergencyThread(), "线程一");
5          t.start();
6
7          for(int i = 1; i < 6; i++){
8              System.out.println(Thread.currentThread().getName()+"输入" + i);
9              if(i == 2){
10                 try{
11                     t.join();//t对应的线程在此插队， 当前线程阻塞。
12                 }catch(InterruptedException e){
13                     e.printStackTrace();
14                 }
15             }
16             try{
17                 Thread.sleep(500);//线程休眠500毫秒， 即半秒
18             }catch(InterruptedException e){
19                 e.printStackTrace();
20             }
21         }
22     }
23 }
24 class EmergencyThread implements Runnable
25 {
26     public void run(){
27         for(int i = 1; i < 6; i++){
28             System.out.println(Thread.currentThread().getName()+"输入"+i);
29             try{
30                 Thread.sleep(500);//线程休眠500毫秒， 即半秒
31             }catch(InterruptedException e){
32                 e.printStackTrace();
33             }
34         }
35     }
36 }
```

注意线程做这些操作的时候,需捕捉异常

```
main输入1
线程一输入1
main输入2
线程一输入2
线程一输入3
线程一输入4
线程一输入5
main输入3
main输入4
main输入5
```

复习


第五次课的内容
进程
线程
并发
概念
继承Thread
实现Runnable
区别
创建线程
线程状态
新建
就绪
阻塞
运行
死亡
线程调度
优先级
休眠
让步
插队