线程

2.某楼盘摇号买房，分为普通号和VIP号，50个普通号，10个VIP号。VIP号的选房时间为普通号的2倍，开始普通号和VIP号并行叫号，叫到VIP号的概率比普通号更高，当普通号叫完第10号时，要求先让VIP号全部选完，再让普通号选房，用多线程模拟这个过程。

```
package com.southwind.thread;

public class Test {
    public static void main(String[] args) {
        //VIP选房的线程
//      VIPRunnable vip = new VIPRunnable();
//      Thread thread = new Thread(vip);
//      thread.setPriority(Thread.NORM_PRIORITY+3);
//      thread.start();
        VIPThread vip = new VIPThread();
        vip.setPriority(Thread.MAX_PRIORITY);
        vip.start();

        //普通号选房的线程
        NormalRunnable normal = new NormalRunnable();
        normal.setThread(vip);
        Thread thread2 = new Thread(normal);
        thread2.setPriority(Thread.MIN_PRIORITY);
        thread2.start();
    }
}

class VIPThread extends Thread{
    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 1; i <= 10; i++) {
            System.out.println("VIP"+i+"正在选房");
            try {
                Thread.currentThread().sleep(2000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

//class VIPRunnable implements Runnable{
//
//  @Override
```

```
//  public void run() {
//      // TODO Auto-generated method stub
//      for (int i = 1; i <= 10; i++) {
//          System.out.println("VIP"+i+"正在选房");
//          try {
//              Thread.currentThread().sleep(2000);
//          } catch (InterruptedException e) {
//              // TODO Auto-generated catch block
//              e.printStackTrace();
//          }
//      }
//  }
//
//}

class NormalRunnable implements Runnable{
    private Thread thread;

    public Thread getThread() {
        return thread;
    }

    public void setThread(Thread thread) {
        this.thread = thread;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 1; i <= 50; i++) {
            if(i == 10) {
                try {
                    thread.join();
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
            }
            System.out.println("普通号"+i+"正在选房");
            try {
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

}
```

3.用多线程模拟网络购票，“游客”，“学生”，“代理”共同抢15张票。

```
package com.southwind.thread;

public class Test2 {
    public static void main(String[] args) {
        TicketRunnable ticketRunnable = new TicketRunnable();
        Thread t1 = new Thread(ticketRunnable,"游客");
        Thread t2 = new Thread(ticketRunnable,"学生");
        Thread t3 = new Thread(ticketRunnable,"代理");
        t1.start();
        t2.start();
        t3.start();
    }
}

class TicketRunnable implements Runnable{
    //剩余票数
    public int count = 15;
    //已售出的票数
    public int num = 0;
    @Override
    public void run() {
        // TODO Auto-generated method stub
        while(count > 0) {
            try {
                Thread.currentThread().sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            synchronized (TicketRunnable.class) {
                if(count == 0) {
                    return;
                }
                count--;
                num++;
                System.out.println(Thread.currentThread().getName()+"抢到了
第"+num+"张票，还剩"+count+"张票");
            }

        }

    }

}
```

```java
package com.southwind.thread;

public class Test3 {
    public static void main(String[] args) {
        TicketThread t1 = new TicketThread();
        TicketThread t2 = new TicketThread();
        TicketThread t3 = new TicketThread();
        t1.setName("游客");
        t2.setName("学生");
        t3.setName("代理");
        t1.start();
        t2.start();
        t3.start();
    }
}

class TicketThread extends Thread{
    //剩余票数
    public static int count = 15;
    //已售出的票数
    public static int num = 0;
    @Override
    public void run() {
        // TODO Auto-generated method stub
        while(num < 15) {
            try {
                Thread.currentThread().sleep(500);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            synchronized (TicketThread.class) {
                if(num == 15) {
                    return;
                }
                count--;
                num++;
                System.out.println(Thread.currentThread().getName()+"抢到了第"+num+"张票，还剩"+count+"张票");
            }
        }
    }
}
```

wait：线程等待

wait的功能和sleep类似，都是让线程暂停执行任务，但是其实是两个完全不同的方法。

sleep是Thread类中的方法，让当前线程实例对象暂停执行任务，进入阻塞状态。

wait是Object类的方法，所以它不是针对线程对象的方法，而是针对线程对象要访问的资源对象的方法。

即调用A对象的wait方法表示：让当前正在访问A对象的线程暂停，同时它有一个前提，当前线程对象必须拥有A对象，所以wait方法只能在同步方法或同步代码块中使用，否则会抛出IllegalMonitorStateException异常。

```
package com.southwind.thread;

public class Test4 {
    public static void main(String[] args) {
        A a = new A();
        new Thread(new Runnable() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                for (int i = 0; i < 10; i++) {
                    a.test(i);
                }
            }
        }).start();
    }
}

class A {
    public synchronized void test(int i) {
        if(i == 5) {
            try {
                this.wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        try {
            Thread.currentThread().sleep(1000);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        System.out.println(i+"---A");
    }
}
```

如何解除wait造成的阻塞?

1.指定wait时间，调用wait(long millis)即可，millis毫米之后会自动解除阻塞，和sleep(long millis)类似的方法。

2.notify：唤醒线程

```
package com.southwind.thread;

public class Test4 {
    public static void main(String[] args) {
        A a = new A();
        new Thread(new Runnable() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                for (int i = 0; i < 10; i++) {
                    a.test(i);
                }
            }
        }).start();

        new Thread(new Runnable() {

            @Override
            public void run() {
                // TODO Auto-generated method stub
                try {
                    Thread.currentThread().sleep(10000);
                } catch (InterruptedException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                a.test2();
            }
        }).start();
    }
}

class A {
    public synchronized void test(int i) {
        if(i == 5) {
            try {
                this.wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        try {
```

```
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            System.out.println(i+"---A");
        }
        public synchronized void test2() {
            this.notify();
        }
    }
```

生产者消费者模型

```
package com.southwind.producecustomer;

public class Hamburger {
    private int id;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
    public Hamburger(int id) {
        this.id = id;
    }

    @Override
    public String toString() {
        return "Hamburger [id=" + id + "]";
    }

}
```

```
package com.southwind.producecustomer;
/*
 * 栈：后进先出
 */
public class SyncStack {
    public Hamburger[] array = new Hamburger[6];
    public int index = 0;
    /*
```

```
     * 向容器中添加汉堡
     */
    public synchronized void push(Hamburger hamburger) {
        while(index == array.length) {
            try {
                this.wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        this.notify();
        array[index] = hamburger;
        index++;
        System.out.println("生产了一个汉堡: "+hamburger);
    }

    /*
     * 从容器中取出汉堡
     */
    public synchronized Hamburger pop() {
        while(index == 0) {
            try {
                this.wait();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        this.notify();
        index--;
        System.out.println("消费了一个汉堡: "+array[index]);
        return array[index];
    }
}
```

```
package com.southwind.producecustomer;

public class Produce implements Runnable{
    private SyncStack syncStack = null;
    public Produce(SyncStack syncStack) {
        // TODO Auto-generated constructor stub
        this.syncStack = syncStack;
    }
    @Override
    public void run() {
```

```
        // TODO Auto-generated method stub
        for (int i = 0; i < 20; i++) {
            Hamburger hamburger = new Hamburger(i);
            this.syncStack.push(hamburger);
            try {
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}
```

```
package com.southwind.producecustomer;

public class Consumer implements Runnable{
    private SyncStack syncStack;

    public Consumer(SyncStack syncStack) {
        this.syncStack = syncStack;
    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        for (int i = 0; i < 20; i++) {
            this.syncStack.pop();
            try {
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }

}
```

```
package com.southwind.producecustomer;

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

```
        SyncStack syncStack = new SyncStack();
        Produce produce = new Produce(syncStack);
        Consumer consumer = new Consumer(syncStack);
        new Thread(produce).start();
        new Thread(produce).start();
        new Thread(produce).start();
        new Thread(consumer).start();
        new Thread(consumer).start();
    }
}
```