

实验报告三

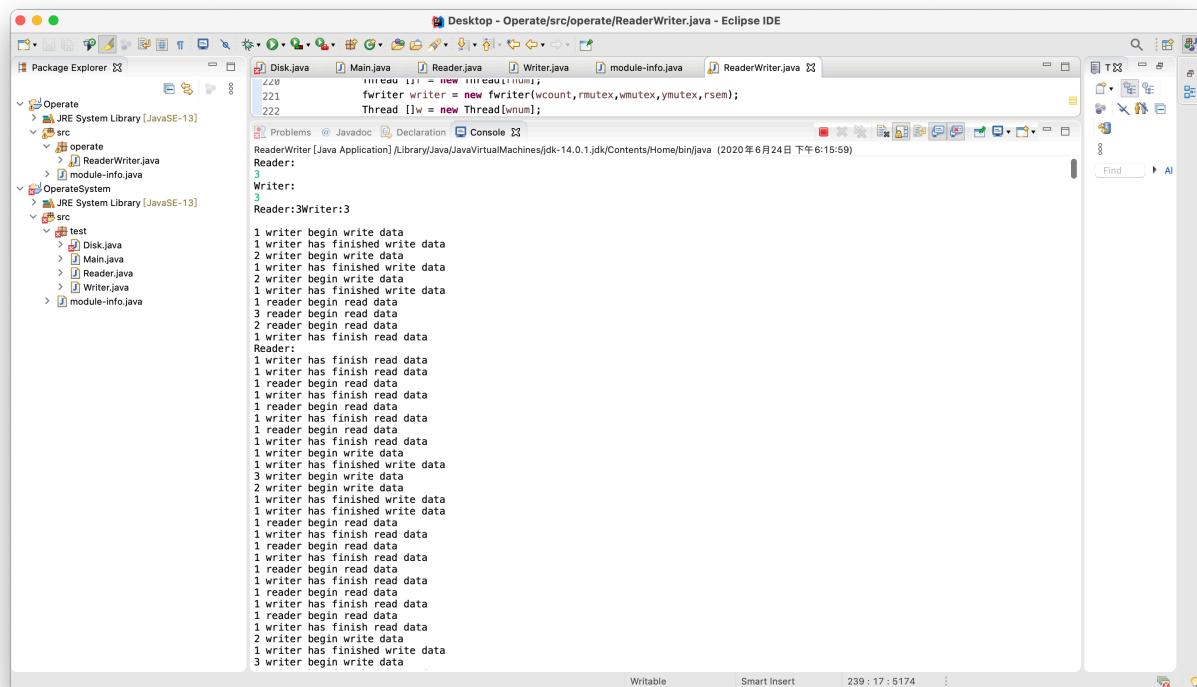
题目：用JAVA语言的同步方法解决读者/写者问题，要求写者优先

目的：认识读写问题中写优先的合理性，能采用同步方法实现。

要求：编写一个含有同步方法的类，其中含有四个同步方法start_read, finish_read, start_write, finish_write。

说明：读者使用方法：{start_read, 读操作, finish_read}; 写者使用方法：{start_write, 写操作, finish_write}。创建若干个读者和若干个写者，执行上述操作，并输出当前访问数据的读者或写者数量。

结果：



The screenshot shows the Eclipse IDE interface with the following details:

- Project Structure:** The Package Explorer view shows a project named "Operate" with packages "src" and "operate". The "src" package contains "Disk.java", "Main.java", "Reader.java", "Writer.java", and "module-info.java". The "operate" package contains "ReaderWriter.java".
- Code Editor:** The code for `ReaderWriter.java` is visible, showing the implementation of Reader and Writer classes with their respective start and finish methods.
- Console Output:** The Console view displays the execution results. It shows a sequence of log messages indicating the start and finish of read and write operations for three threads (Reader: 1, Writer: 2, Writer: 3). The log messages are:

```
1 writer begin write data
1 writer has finished write data
2 writer begin write data
1 writer has finished write data
2 writer begin write data
1 writer has finished write data
1 reader begin read data
3 reader begin read data
1 reader begin read data
2 reader begin read data
1 writer has finish read data
Reader:
1 writer has finish read data
1 writer has finish read data
1 reader begin read data
1 writer has finish read data
1 reader begin read data
1 writer has finish read data
1 reader begin read data
1 writer has finish read data
1 writer begin write data
1 writer has finished write data
3 writer begin write data
2 writer begin write data
1 writer has finished write data
1 writer has finished write data
1 writer begin write data
1 reader begin read data
1 writer has finish read data
1 writer begin write data
2 writer begin write data
1 writer has finished write data
3 writer begin write data
```

代码：

```
package operate;
import java.util.*;

class Semaphore{
    int value;
    public Semaphore(int v){
        this.value = v;
    }
    public synchronized void p(){
        value = value-1;
        if(value<0){
            try {
                wait();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    public synchronized void v(){
        value = value+1;
        if(value<=0){
            this.notifyAll();
        }
    }
}

public Reader(int c,Semaphore r,Semaphore w){
    this.count = c;
    this.rmutex = r;
    this.wmutex = w;
}
public void run(){
    while(true){
        rmutex.p();
        if(count == 0) wmutex.p();
        count++;
        rmutex.v();
        System.out.println(count+" begin read data");
        rmutex.p();
        count--;
        System.out.println("1 reader has finished reading data");
        if(count == 0) wmutex.v();
        rmutex.v();
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
}
```

```

class Writer implements Runnable{
    Semaphore wmutex;
    int wnumber;
    public Writer(Semaphore w){

        this.wmutex = w;
    }
    public void run(){
        while(true){
            wmutex.p();
            System.out.println("writer "+"begin write data");
            System.out.println("1 writer finish write data");
            wmutex.v();
            try {
                Thread.sleep((long) (Math.random()*1000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

82 class freader implements Runnable{
83     int count;
84     int rnumber;
85     int z;
86     Semaphore rmutex;
87     Semaphore wmutex;
88     Semaphore zmutex;
89     Semaphore rsem;
90     public freader(int c,Semaphore r,Semaphore w,Semaphore z,Semaphore z1){
91         this.count = c;
92         this.rmutex = r;
93         this.wmutex = w;
94         this.zmutex = z;
95         this.rsem = z1;
96     }
97     public void run(){
98         while(true){
99             zmutex.p();
100            rsem.p();
101            rmutex.p();
102            count++;
103            if(count == 1) wmutex.p();
104            wmutex.v();
105            rmutex.v();
106            rsem.v();
107            zmutex.v();
108            System.out.println(count+" reader"+" begin read data");
109            rmutex.p();
110            count--;
}

```

```

111     System.out.println("1 writer has finish read data");
112     if(count == 0) wmutex.v();
113     rmutex.v();
114     try {
115         Thread.sleep((long) (Math.random()*1000));
116     } catch (InterruptedException e) {
117         e.printStackTrace();
118     }
119 }
120 }
121 }
122 }
123 class fwriter implements Runnable{
124     int wcount;
125     Semaphore rmutex;
126     Semaphore wmutex;
127     Semaphore ymutex;
128     Semaphore rsem;
129     int wnumber;
130     public fwriter(int c,Semaphore r,Semaphore w,Semaphore y,Semaphore z1){
131         this.rmutex = r;
132         this.wmutex = w;
133         this.ymutex = y;
134         this.rsem = z1;
135     }
136 }
137
138 public class ReaderWriter{
139     public static void main(String[] args){
140         while(true){
141             Scanner sc = new Scanner(System.in);
142             System.out.println("Reader:");
143             int rnum = sc.nextInt();
144
145             if(rnum<0|rnum>10){
146                 System.out.println("Error!");
147                 continue;
148             }
149             System.out.println("Writer:");
150             int wnum = sc.nextInt();
151             System.out.println("Reader:"+rnum+"Writer:"+wnum);
152             if(wnum<0|wnum>10){
153                 System.out.println("Error!");
154                 continue;
155             }
156             System.out.println(" ");
157             int first = 2;
158             if(first<1|first>2){
159                 System.out.println("Error!");
160                 continue;
161             }
162         }
163     }
164 }
```

```

if(first==1){
    int count = 0;
Semaphore rmutex = new Semaphore(1);
Semaphore wmutex = new Semaphore(1);
Reader reader = new Reader( count,rmutex,wmutex);
Thread []r = new Thread[rnum];
Writer writer = new Writer(wmutex);
Thread []w = new Thread[wnum];
int b[] = new int[wnum];
for(int i=0;i<rnum;i++){
    int c = 0;
    r[i] = new Thread(reader);
    r[i].start();
    c++;
}
for(int dcount=0;dcount<wnum;dcount++){
    int d = 0;
    b[d]=dcount+1;
    w[dcount] = new Thread(writer);
    w[dcount].start();
    d++;
}
}

if(first==2){
    int count = 0;
    int wcount = 0;
Semaphore rmutex = new Semaphore(1);
Semaphore wmutex = new Semaphore(1);
Semaphore ymutex = new Semaphore(1);
Semaphore rsem = new Semaphore(1);
Semaphore zmutex = new Semaphore(1);
freader reader = new freader( count,rmutex,wmutex,zmutex,rsem);
Thread []r = new Thread[rnum];
fwriter writer = new fwriter(wcount,rmutex,wmutex,ymutex,rsem);
Thread []w = new Thread[wnum];
int b[] = new int[wnum];
for(int dcount=0;dcount<wnum;dcount++){
    int d = 0;
    b[d]=dcount+1;
    w[dcount] = new Thread(writer);
    w[dcount].start();
    d++;
}
for(int i=0;i<rnum;i++){
    int c = 0;
    r[i] = new Thread(reader);
    r[i].start();
    c++;
}
}

```

