In today's lecture I will talk about three programs from the text (chap 15) on Threading.

The first deals with the creating, starting, and putting threads to sleep:

```java
public class ThreadTester {
    public static void main(String args[]) {
        PrintThread thread1, thread2, thread3, thread4;
        thread1 = new PrintThread("threadA");
        thread2 = new PrintThread("threadB");
        thread3 = new PrintThread("threadC");
        thread4 = new PrintThread("threadD");

        System.out.println("Starting threads");
        thread1.start();
        thread2.start();
        thread3.start();
        thread4.start();
        System.out.println("Threads started");
    }
}
```

```java
class PrintThread extends Thread {
    private int sleepTime;

    public PrintThread (String name) {
        super(name);
        sleepTime = (int) (10000* Math.random());
        System.out.println("Name: " + getName() + "; sleep: " + sleepTime);
    }

    public void run() {
        try {
            System.out.println(getName() + " going to sleep");
            Thread.sleep(sleepTime);
        }
        catch( InterruptedException exception) {
            System.out.println(exception.toString());
        }
    }
}
```
Typical output here would be

```
Z:\11Pic20\Lecture_20>java ThreadTester
Name: Thread-0; sleep: 7420
Name: Thread-1; sleep: 8754
Name: Thread-2; sleep: 9330
Name: Thread-3; sleep: 3064

Starting threads

Threads started

Thread-0 going to sleep
Thread-1 going to sleep
Thread-2 going to sleep
Thread-3 going to sleep
Thread-3 done sleeping
Thread-0 done sleeping
Thread-1 done sleeping
Thread-2 done sleeping
```

The next program deals with what can happen when threads are not synchronized. There are four different programs. The first is a class that contains a variable `sharedInt`, along with two methods for manipulating this variable:

```java
public class HoldIntegerUnsynchronized
{
    private int sharedInt = -1;

    public void setSharedInt(int val)
    {
        System.err.println(
                        Thread.currentThread().getName() + 
                        " setting SharedInt to " + val);
        sharedInt = val;
    }

    public int getSharedInt()
    {
        System.err.println(
                        Thread.currentThread().getName() + 
                        " gotten SharedInt = " + sharedInt);
        return sharedInt;
    }
}
```
" retrieved SharedInt " + sharedInt);
    return sharedInt;
}
} // end getSharedInt()
} // end class HoldIntegerUnsynchronized

The next two are threads. The first one ProduceInteger, assigns values to the variable setSharedInt of the previous program

public class ProduceInteger extends Thread
{
    private HoldIntegerUnsynchronized pHold;

    public ProduceInteger(HoldIntegerUnsynchronized h)
    {
        super("Produce integer");
        pHold = h;
    }

    public void run()
    {
        for (int count = 1; count <= 10; count++)
        {
            try
            {
                Thread.sleep( (int)(5000*Math.random()));
            }
            catch(InterruptedException e)
            {
                System.err.println(e.toString());
            }
            pHold.setSharedInt(count);
        }
        System.err.println(getName() + " finished producing values" + "\n Terminating " + getName() );
    }
} // end run()
} // end class ProduceInteger

The next gets values of SharedInt:

public class ConsumeInteger extends Thread
{
    private HoldIntegerUnsynchronized cHold;

    public ConsumeInteger( HoldIntegerUnsynchronized h)
    {
        super("Consume Integer");
        cHold = h;
    }
} // end class ConsumeInteger()
public void run()
{
    int val, sum = 0;

    do
    {
        try
        {
            Thread.sleep((int) ( Math.random()*5000 ) );
        }
        catch(InterruptedException e)
        {
            System.err.println(e.toString());
        }

        val = cHold.getSharedInt();
        sum += val;
    } // end do
    while (val != 10);

    System.err.println(getName() +
                        " retrieved values totalling:
                        "+sum+
                        "\nTerminating " + getName() );

} // end run()
// end class ConsumeInteger

The last is a driver:

public class SharedCell
{
    public static void main(String args[])
    {
        HoldIntegerUnsynchronized h =
        new HoldIntegerUnsynchronized();

        ProduceInteger p = new ProduceInteger(h);
        ConsumeInteger c = new ConsumeInteger(h);

        p.start();
        c.start();

    } // end main(String args[])

} // end class SharedCell

Typical output is:
The next set of four programs deal with synchronizing the getting and setting of the sharedInt. The critical part occurs in the class HoldIntegerSynchronized, which introduces the idea of a synchronized method, as well as a boolean variable writeable that controls the running of the threads.

```java
public class HoldIntegerSynchronized {
    private int sharedInt = -1;
    private boolean writeable = true;

    public synchronized void setSharedInt(int val) {
        while(!writeable)
        {
            try
            {
            
```
wait();
}
catch(InterruptedException e)
{
    e.printStackTrace();
}
} // end while()

System.err.println(
    Thread.currentThread().getName() + 
    " setting SharedInt to " + val);
sharedInt = val;
writeable = false;
notify();

} // end setSharedInt(int val)

public synchronized int getSharedInt()
{
    while (writeable)
    {
        try
        {
            wait();
        }
        catch(InterruptedException e)
        {
            e.printStackTrace();
        }
    } //end while()

    writeable = true;
    notify();

    System.err.println( 
        Thread.currentThread().getName() + 
        " retrieved SharedInt " + sharedInt);
    return sharedInt;

} // end getSharedInt()

} // end class HoldIntegerUnsynchronized

The changes in the three remaining classes are simple variations that reflect the above class.

public class ProduceInteger extends Thread
{
    private HoldIntegerSynchronized pHold;

    public ProduceInteger(HoldIntegerSynchronized h)
    {
        super("Produce integer");
    }
```java
public void run()
{
    for (int count = 1; count <= 10; count++)
    {
        try {
            Thread.sleep((int)(5000*Math.random()));
        } catch(InterruptedException e) {
            System.err.println(e.toString());
        }

        pHold.setSharedInt(count);
    }
    System.err.println(getName() + "finished producing values" + "\n Terminateing " + getName());
}
```

```java
public class ConsumeInteger extends Thread
{
    private HoldIntegerSynchronized cHold;

    public ConsumeInteger( HoldIntegerSynchronized h)
    {
        super("Consume Integer");
        cHold = h;
    }

    public void run()
    {
        int val, sum = 0;
        do
        {
            try {
                Thread.sleep((int) ( Math.random()*5000) );
            } catch(InterruptedException e) {
                System.err.println(e.toString());
            }
            val = cHold.getSharedInt();
            sum += val;
        } while (val != 10);
    }
}
```
public class SharedCell{
    public static void main(String args[])
    {
        HoldIntegerSynchronized h =
        new HoldIntegerSynchronized();

        ProduceInteger p = new ProduceInteger(h);
        ConsumeInteger c = new ConsumeInteger(h);

        p.start();
        c.start();
    }
}

The output in this case is synchronized:
Z:\11Pic20\Lecture_20>java SharedCell
Consume Integer retrieved SharedInt -1
Consume Integer retrieved SharedInt -1
Consume Integer retrieved SharedInt -1
Produce integer setting SharedInt to 1
Produce integer setting SharedInt to 2
Consume Integer retrieved SharedInt 2
Produce integer setting SharedInt to 3
Consume Integer retrieved SharedInt 3
Consume Integer retrieved SharedInt 3
Produce integer setting SharedInt to 4
Produce integer setting SharedInt to 5
Consume Integer retrieved SharedInt 5
Consume Integer retrieved SharedInt 5
Produce integer setting SharedInt to 6
Consume Integer retrieved SharedInt 6
Consume Integer retrieved SharedInt 6
Produce integer setting SharedInt to 7
Consume Integer retrieved SharedInt 7
Consume Integer retrieved SharedInt 7
Produce integer setting SharedInt to 8
Produce integer setting SharedInt to 9
Consume Integer retrieved SharedInt 9
Consume Integer retrieved SharedInt 9
Produce integer setting SharedInt to 10
Produce integer finished producing values
   Terminating Produce integer
Consume Integer retrieved SharedInt 10
Consume Integer retrieved values totalling: 69
Terminating Consume Integer

Z:\11Pic20\Lecture_20>