

Threads Rethreaded

Execution of Synchronized Sections

- Thread A enters synch section
 1. A checks for lock; points lock to self
 2. Enters code segment
 3. Completes code segment
 4. Releases lock (points it back to nobody)
- Thread B enters synch at 3
 - B checks for lock; can't get it; blocks
 - When A completes 4, B can get lock
 - B repeats 1-4 itself

Synchronization Examples

```
public class MyClass {
    private Vector _myVector=new Vector();
    private static _myList=new LinkedList();
    private List _mySList=new SortedList();
    public synchronized void writeData(int e) {
        _myVector.add(new Integer(e));
        _myList.add(new Integer(e));
        _mySList.add(new Integer(e));
    }
    private static void writeSData(int e) {
        _myList.add(new Integer(e));
    }
    private Integer readData(e) {
        return (Integer)_mySList.get(new Integer(e));
    }
}
```

Synchronization Examples

```
public class MyClass {
    private Vector _myVector=new Vector();
    private static _myList=new LinkedList();
    private List _mySList=new SortedList();
    public synchronized void writeData(int e) {
        _myVector.add(new Integer(e));
        _myList.add(new Integer(e));
        _mySList.add(new Integer(e));
    }
    private static void writeSData(int e) {
        _myList.add(new Integer(e));
    }
    private synchronized Integer readData(e) {
        return (Integer)_mySList.get(new Integer(e));
    }
}
```

Synchronization Examples

```
public class MyClass {
    private Vector _myVector=new Vector();
    private static _myList=new LinkedList();
    private List _mySList=new SortedList();
    public void writeData(int e) {
        _myVector.add(new Integer(e));
        _myList.add(new Integer(e));
        _mySList.add(new Integer(e));
    }
    private synchronized static void
        writeSData(int e) {
        _myList.add(new Integer(e));
    }
    private Integer readData(e) {
        return (Integer)_mySList.get(new Integer(e));
    }
}
```

Synchronization Examples

```
public class MyClass {
    private Vector _myVector=new Vector();
    private static _myList=new LinkedList();
    private List _mySList=new SortedList();
    public void synchronized writeData(int e) {
        _myVector.add(new Integer(e));
        _myList.add(new Integer(e));
        _mySList.add(new Integer(e));
    }
    private synchronized static void
    writeSData(int e) {
        _myList.add(new Integer(e));
    }
    private synchronized Integer readData(e) {
        return (Integer)_mySList.get(new Integer(e));
    }
}
```

Synchronization Examples

```
public class MyClass {
    private Vector _myVector=new Vector();
    private static _myList=new LinkedList();
    private List _mySList=new SortedList();
    public void writeData(int e) {
        _myVector.add(new Integer(e));
        synchronized (_myList) {
            _myList.add(new Integer(e)); }
        _mySList.add(new Integer(e));
    }
    private static void writeSData(int e) {
        synchronized (_myList) {
            _myList.add(new Integer(e)); }
    }
    private Integer readData(e) {
        return (Integer)_mySList.get(new Integer(e));
    }
}
```

Returning to the Thread Corrup. Ex.

```
private int[] _multArray[] = new int[128];
private constructor() {
    for (int i=0;i<_multArray.length;++i) {
        _multArray[i]=i;
    }
}
private void _seedArray(int s) {
    for (int i=0;i<_multArray.length;++i) {
        _multArray[i]=i*s;
    }
}
private void _writeToFile() {
    _myFile.print("_multArray=[ ");
    for (int i=0;i<_multArray.length;++i) {
        _myFile.print(_multArray[i] + " ");
    }
    _myFile.println("]");
}
```

Creating Threads in Java

- Every thread is *represented* with an object of type `Thread`
- This object gives you a handle to manipulate the thread -- start it, kill it, prioritize it, etc.
- `Thread` implements the interface `Runnable`
- `Runnable` specifies only one method:
 - `public void run()`
- When a thread is started, it enters the `run()` method. When `run()` returns, the thread ends (dies).
- `run()` can call any other method it likes, `sleep`, etc.

Two Ways to Make Threads

- To make a new thread, need two things:
 - A `run()` method
 - Start the thread running
- To get a `run()` method, either
 1. Subclass `Thread` and override `run()`, or
 2. Make another class that implements `Runnable` itself
- To start a new thread running, either
 1. Call `Thread.start()`
 2. `Thread thr=new Thread(runnableObj)`

Example of Method 1

```
class myThreadClass extends Thread {  
    public void run() {  
        System.out.println("Here goes the thread...");  
    }  
}
```

```
// elsewhere...
```

```
public void someOtherMethod() {  
    Thread thrObj=new myThreadClass();  
    thrObj.start();  
}
```

Example of Method 2

```
class myRunnable implements Runnable {  
    public void run() {  
        System.out.println("Here goes the thread...");  
    }  
}
```

```
// elsewhere...
```

```
public void someOtherMethod() {  
    Runnable runObj=new myRunnable();  
    Thread thrObj=new Thread(runObj);  
    thrObj.start();  
}
```

Method 2 + Anonymous Inner Classes

```
public void miscMethod() {  
    Thread thrObj=new Thread(new Runnable {  
        public void run() {  
            System.out.println("In new thread...");  
        }  
    }).start();  
}
```