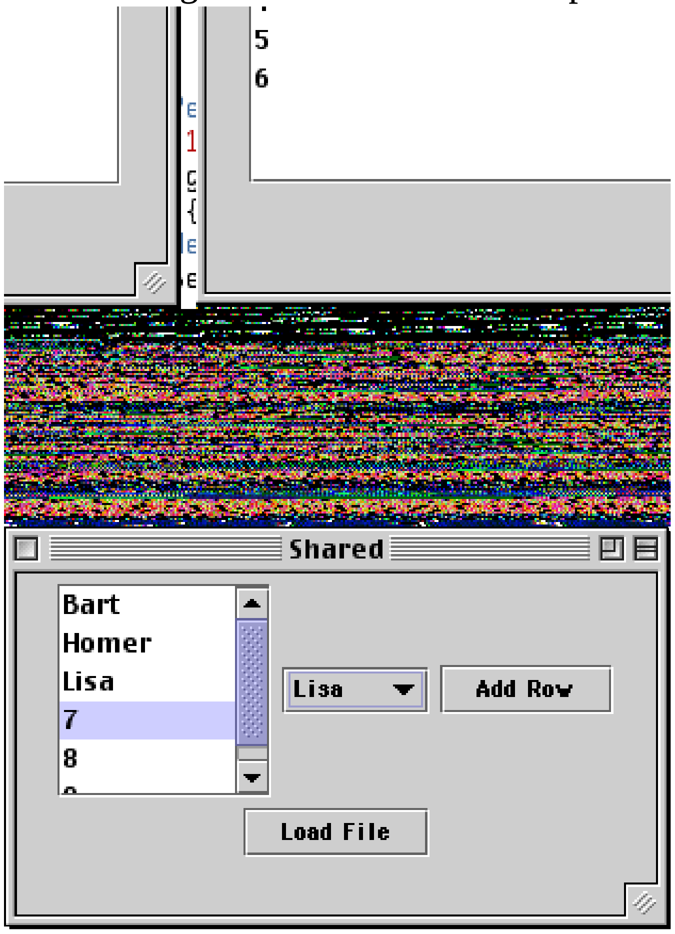
CS193k, Stanford
Spring, 2000-01

Handout #3
Nick Parlante

Swing 3 + Threads

List Example

Another MVC built-in class
Show using one model with multiple views



MyListModel Code

```
// MyListModel.java
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

import java.awt.event.*;
import java.util.*;  // for Vector

import javax.swing.event.*;

import java.io.*; // for File
```

```
/*
 A simple example of implemention a ListModel.
 In this case, we just use Vector.
 AbstractListModel keeps track of the listeners for us, but
 we still need to trigger the notifications.
*/
class MyListModel extends AbstractListModel {

   private Vector data = new  Vector();

   // Must override these two from ListModel
   public int getSize() {
      return(data.size());
   }

   public Object getElementAt(int index) {
      return(data.elementAt(index));   // could sanity check index
   }


   // My methods so clients can add and remove elements
   // on the data model (clients can also use the standard
   // getElementAt() for accessing).
   public int addRow(String string) {
      data.addElement(string);

      // Must send the following
      // (AbstractListModel provides the listener support for us)
      fireIntervalAdded(this, data.size()-1, data.size()-1);
      return(data.size()-1);
   }

   public void deleteRow(int row) {
      // ??? could error check the row int
      data.removeElementAt(row);
      fireIntervalRemoved(this, row, row);
   }

   // If we had an operation that CHANGED the contents of a row, we would
   // fireContentsChanged(this, row, row)

   static int count = 0;

   // Demonstrate MyListModel
   public static void doList() {
      final JFrame frame = new JFrame("List");
      final Container container = frame.getContentPane();
      container.setLayout(new FlowLayout());

      final MyListModel listModel = new MyListModel();
      final JList list = new JList(listModel);
      JScrollPane scrollpane = new JScrollPane(list);

      // could do this, or use the medium default size
      // scrollpane.setPreferredSize(new Dimension(200,120));

      // Button to add a row to the model
      JButton button = new JButton("Add Row");
      container.add(button);
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               count++;
               int newRow = listModel.addRow(Integer.toString(count));
               list.setSelectedIndex(newRow);
            }
         }
      );


      // Delete the currently selected row
      final JButton button2 = new JButton("Delete Row");
      container.add(button2);
      button2.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               // can return -1
               int sel = list.getSelectedIndex();
               if (sel != -1) {
                  listModel.deleteRow(sel);
                  list.clearSelection();

               }
            }
         }
      );

      container.add(scrollpane);
      frame.setVisible(true);

   }


   // Demonstrate using a model for a list and a combobox
   public static void doShared() {

      final JFrame frame = new JFrame("Shared");
      final Container container = frame.getContentPane();
      container.setLayout(new FlowLayout());


      // DefaultComboBoxModel has basic storage built in
      // -- has getSize() and addElement()
      // DefaultComboBoxModel implements ComboBoxModel
      // ComboBoxModel is a subclass of ListModel

      final DefaultComboBoxModel model = new DefaultComboBoxModel();
      String[] strings = {"Bart", "Homer", "Lisa"};

      for(int i=0; i<strings.length; i++) {
         model.addElement(strings[i]);
      }

      final JList list = new JList(model);
      JScrollPane scrollpane = new JScrollPane(list);
      scrollpane.setPreferredSize(new Dimension(100,100));
      container.add(scrollpane);
      list.getSelectionModel().
         setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

      JComboBox combo = new JComboBox(model);
      container.add(combo);


      // Add a row the model
      JButton button = new JButton("Add Row");
      container.add(button);
      button.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               count++;
               model.addElement(Integer.toString(count));

            }
         }
      );

      // Read the lines out of a file and add them
      JButton button2 = new JButton("Load File");
      container.add(button2);
      button2.addActionListener(
         new ActionListener() {
            public void actionPerformed(ActionEvent e) {
               JFileChooser chooser = new JFileChooser(".");
               int status = chooser.showOpenDialog(frame);
               if (status == JFileChooser.APPROVE_OPTION) {
                  File file = chooser.getSelectedFile();

                  try {
                     FileReader fileReader = new FileReader(file);

                     // The buffered layer is optional by recommended
                        Reader bufferedReader = new BufferedReader(fileReader);

                        StreamTokenizer tokenizer = new
StreamTokenizer(bufferedReader);

                     // Try to set the tokenizer to reader "words" line by line
                     //tokenizer.resetSyntax();
                     //tokenizer.whitespaceChars( '\n', '\r');    // no

                     //tokenizer.ordinaryChar(' ');   // no
                     //tokenizer.ordinaryChar('\t');

                     //tokenizer.wordChar(' '); // no such method
                     tokenizer.wordChars(' ', ' ');   // now each line counts as a
token
                     tokenizer.wordChars('\t', '\t');
                     int tok;

                     // The standard loop to get all the tokens in a file
                     while ((tok = tokenizer.nextToken()) != StreamTokenizer.TT_EOF)
{
                        model.addElement(tokenizer.sval);
                     }
                  }
                  catch (IOException ignored) {
                  }
               }
            }
         }
      );
      frame.setVisible(true);
   }

   public static void main(String args[]) {
      doList();
      doShared();
   }

}
```

Faster Computers

Why faster?

How is it that computers are faster now than 10 years ago?

a. Process improvements -- chips are smaller and run faster

b. Superscalar pipelining parallelism techniques -- doing more than one thing at a time from the one instruction stream.

Instruction Level Parallelism -- limit of 3-4x

We are well in to the diminishing-returns region of ILP technology.

100 million transistors

Suppose you have a chip with 100 million transistors

What will you do with them all?

Extract more ILP? -- not really

More and bigger cache -- ok, but there are limits

Explicit concurrency -- YES

Explicit Concurrency

Chip

The chip(s) can support multiple threads.

Software

The software must be coded to use multiple threads -- this is a significant cost, but we're getting better at it.

CPU Concurrency Trends

1. Multiple CPU's
2. "Multiple cores" on one chip
   - They can share on-chip L1 cache as well
   - A goo d way to use up more transistors, without doing a whole new design.
3. Chip Multi-threading
   - One core with multiple sets of registers
   - The core shifts between one thread / register-set and another quickly -- say whenever there's an L1 miss.
   - Neat feature: hide the latency by overlapping a few active threads.

Threading

- Thread level vs. Process Level
- Threads share address space
- OS's now support "inexpensive" threads -- on the order of 10-50 per process
- Separate processes are heavyweight -- separate address space, large start-up cost

Multiple processors

- CPU intensive could get value from extra processor (but why code in Java for CPU bound problem?)
- Memory intensive less so
- Disk / Network intensive even less so

Network / Disk -- Hide The Latency

- Use threads to efficiently block when data is not there
- Even with one CPU, can get excellent results
- Suppose very fast CPU, and very slow network -- even with coarse locking, may get excellent results. The threads are blocked most of the time anyway, so lock contention is not really a problem.
- This is what Java threads are really good for.

Why Concurrency Is Hard

- No language construct can make the problem go away (in contrast to mem management which was made to go away with GC). The programmer must be involved.
- There is no fixed programmer recipe that will just make the problem go away.
- Hard for classes to pass the "clueless client" test -- the client may really need to understand the internal lock model of a class to use it correctly.
- Concurrency bugs are very, very latent. The easiest bugs are the ones that happen every time.

In contrast, concurrency bugs show up rarely, they are very machine, VM, and current machine loading dependent, and as a result they are hard to repeat.

"Concurrency bugs -- the memory bugs of the 21st century."

Rule of thumb: if you see something bizarre happen, don't just pretend it didn't happen. Note the current state as best you can.

Native vs. Green

Thread Implementation

Green = 1 native thread -- easiest to implement

Native = 1 native thread for each Java thread -- most common

Mixed = n native threads for k Java threads

As of Java 1.2, nobody uses Green threads

Coding Strategies

Cooperative "green" threads -- schedule on yield(), sleep(), lock acquire (through system call)

In that case, your code should call yield() every now and then.

Native "preemptive" threads -- threads may be scheduled on above + preemptively

If a program works in green threads, it may still fail with native threads.

Green Reliability

Green threads are less likely to expose concurrency bugs since they do not take away the thread of control in the middle of some statements.

```
{
   i = i +1;   // won't loose it here
   next = a[i];   // or here
   foo();   // maybe here, depending on what foo does
}
```

Java : Compile-Time Locks / Structure

The Java "synchronized" lock acquisition structure is formally structured at compile time.

Structure

```
lock(x) {
  aaa
  bbb
}
```

vs. RT style (not Java)

```
{
  lock(x);
  aaa
  bbb
  unlock(x);
}
```

CT features

Can't mess up the balance -- exceptions, etc. -- always balance
Less flexible

1. Classic Critical Section Problem

```
class Foo {
  int i;

  void incr() {
    i = i + 1;
  }
}
```

2. Java Solution: synchronized

Compile-time
- Part of the source code structure

Acquire the lock on the receiver
- equivalent to synchronized(this)

Errors
- Most common errors derive from loosing track of which lock has been synchronized.

3. Synchronized code

Synch lock on the receiver

```
synchronized void incr() {
  i = i +1;
}
```

Result
- Acquires the lock on **this** -- any other code that uses that lock will block while we're in this section.
- The lock is part of the **receiver** object.

Common Synch Errors

1. Error - must volunteer to be synchronized

```
void decr() {
  i = i -1;
}
```

Only methods that are synchronized are locked out. In this case, decr() can still get in while incr() holds the lock.

2. Error - static methods do not synch on an instance

```
static void incrObj(Foo foo) {
  foo.i = foo.i + 1;
}
```

Solution

Having a static method change the state of an object is weird, but if we ignore that, the solution would be to block on the same lock as the regular synchronized methods...

```
static void incrObj(Foo foo) {
  synchronized(foo) {
    foo.i = foo.i + 1;
  }
}
```

3. Error - Shared Static

```
static int count;
synchronized binky() {
  count = count + 1;
}
```

Problem

binky() will not be running concurrently against one object, but with multiple objects, it could be running concurrently against multiple objects.

a. synch(this) -- same problem

```
void binky() {
  synchronized(this) {
    count = count+1;
  }
}
```

b. synch(lock) -- solution

Add a dedicated lock object used for count...

```
static int count;
static Object countLock = new Object();
void binky() {
  synchronized(countLock) {
    count = count + 1;
  }
}
```

4. Error - Shared Object

```
int[] a;      // suppose all Foo's share a pointer to one a obj
syncronized void binky() {
  a[0] = a[0] + 1;
}
```

Solution

```
void binky() {
  synchronized(a) {
    a[0] = a[0] + 1;
  }
}
```

5. Split Transaction Problem -- Too fine grain

Code

```
class Account {
  int balance;

  public synchronized int getBal() { return(balance); }
  public synchronized void setBal(int val) {balance = val;}
}
```

Problem

Two threads could interleave their calls to get/set just so to get the wrong answer.
The synch is at too fine grain -- the critical section is larger
This is tricky -- the programmer could think "I used synchronized everywhere" so they think it's ok.

Solution

Move the synch out so it covers the whole transaction

```
public synchronized changeBal(int delta) {
  int val = getBal();
  val += delta;
  setBal(val);
}

-or-

public synchronized changeBal(int delta) { balance += delta; }
```