Oops ...

The last example of the last lecture:

```
private static void makeEx (Ex e) {
  e = new Ex();
  e.field = 4;
}
public static void main (String[] args) {
  Ex e = new Ex();  // Should have been just Ex e;
  makeEx(e);
  System.out.println(e.field);
}
```

Comparing objects

Simple comparisons can be done with ==, !=, >=, etc.:

```
int i, j;
... // Give values to i and j.
if (i == j)
  System.out.println ("same");
```

The rules are the same for objects, but they don't work as you'd expect.

```
Ex e = new Ex();
e.field = 2;
Ex f = new Ex();
f.field = 2;
if (e != f)
  System.out.println("not same");
```

The == operator compares the ***values*** of the expressions, and the values here are references.

More object equality tests

You can use the == operator to compare the values of any pair of expressions, as long as the value is what you really want to compare.

```
if (e.field == f.field)
  System.out.println("same");

e = f; // Now they refer to the same object.
if (e == f)
  System.out.println ("same");
```

Comparing strings

Strings are objects. Don't use == to compare them.

```
String line1, line2;
line1 = in.readLine();
line2 = in.readLine();
if (line1.equals(line2))
  ...
```

Use the equals( ) method to see if two strings are the same.

Which string is first?

To find out which string comes first in the alphabet, don't use < and >.
Instead, use the compareTo( ) method, which returns an ***integer***:

```
int line1isInThisDirectionFromLine2
    = line1.compareTo(line2);
```

The value of line1.compareTo(line2) is:

- 0 if they're the same
- < 0 if line1 comes before line2
- > 0 if line1 comes after line2

Use the compareTo( ) method to compare strings.

String comparison: some details

Example:

```
"zoo".compareTo("aardvark") > 0
```

String comparison uses "dictionary order", based on the ***collating sequence***.
Java's collating sequence is Unicode, which includes ASCII.
These are the ***only*** things to remember about Unicode (or ASCII):

- blank comes before all other printable characters
- A .. Z, a .. z, 0 .. 9 are each contiguous and in order.

Washing machines with loads

```
class WashingMachine {
  private String contents;
  public String toString() {
    return "Contents: " + contents;
  }
  public void add (String item) {contents = item;}
}
public class TestWash { ...main... {
    WashingMachine wm = new WashingMachine();
    wm.add("sock");
    wm.add("hat");
    System.out.println(wm);
}}
```

What's the output?

Keeping a list

A Vector object (an object of the Vector class) can hold objects of any class.

```
Vector list = new Vector();
list.addElement("sock");
list.addElement("hat");
```

To get an object back from a Vector:

```
String item = (String) list.elementAt(0);
System.out.println("First item in list: " + item);
```

The cast in front of the elementAt( ) call is essential.

```
item = (String) list.elementAt(1);
System.out.println("Second item in list: " + item);
```

The contents of the list are not changed by calling elementAt( ).

WashingMachine again

```
class WashingMachine {
  private Vector contents;
  public WashingMachine () {
    contents = new Vector();
  }
  public void add (String item) {
    contents.addElement(item);
  }
  public String toString() {
    String result = "Contents: ";
    for (int i = 0; i < contents.size(); i++)
      result += "  "+(String) contents.elementAt(i);
    return result;
  }
}
public class TestWash { ...main... {
    WashingMachine wm = new WashingMachine();
    wm.add("sock");
    wm.add("hat");
    System.out.println(wm);
}}
```