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);
}}
```

Print the middle line

```
import java.io.*;
public class WashingMachine {
  public static void main (String[] a)
      throws IOException {
    BufferedReader in = new BufferedReader(...);
    Vector lines = new Vector();

    final String STOPPER = "quit";
    while (true) {
      String line = in.readLine();
      if (line.equals(STOPPER)) break;
      lines.addElement(line);
    }

    System.out.println("Middle line");
    System.out.println((String)lines.elementAt
      (lines.size()/2));
  }
}
```

Vector rules

Rule 0:
The things stored in a Vector are called ***elements***.
The sequence number of an element in a Vector is called its ***index***.

Rule 1:
The elements are numbered from 0 upwards.

Rule 2:
There are no gaps in the numbering.
When you remove an element, the later ones are renumbered.

Rule 3:
The Vector expands as needed to accommodate newly-added elements.

More Vector rules

<u>Rule 4</u>:
All elements must be objects, but ***any*** type of object is acceptable.

```
Vector v = new Vector();
v.addElement("hi"); // a String element
Student stu = new Student("Jim", "99001");
v.addElement(stu); // a Student element
v.addElement(new WashingMachine());
v.addElement(new File("A:\myData.txt"));
v.addElement(System.in);
```

How can Rule 4 be right?

<u>Question</u>:
How can vectors contain all these different kinds of things?

<u>Answer</u>:
They can't: they only accept objects of one class, namely Object.
 `addElement (Object o) //` the signature of addElement

But … ***every object is an Object!***

More Vector rules

Rule 5:
Elements retrieved from a Vector must be cast to the right type.

Example:
If the element with index 4 is a Student, here's how to get it out of v:

```
Student s = (Student) v.elementAt(4);
```

The compiler knows that v.elementAt(4) is of type Object.
The cast "(Student)" tells it that you're sure it's really also a Student.

Another Rule 4 consequence

Values of primitive types are not allowed in a Vector:

```
v.addElement(5); // NO!
```

To store a primitive value such as an int in a Vector, you have to "wrap" it:

```
int i;
... // code that gives i a value
Integer iWrap = new Integer(i);
v.addElement(iWrap);
... // code that eventually wants the value
Integer getIback = (Integer)v.elementAt(27);
int j = getIback.intValue();
```

Or, more briefly:

```
int j = ((Integer)v.elementAt(27)).intValue();
```

Wrapper classes

Each primitive type corresponds to a ***wrapper class***:

- Integer
- Character
- Double
- Boolean

These classes also hold static methods, some of which you've seen:

- Integer.parseInt
- Double.parseDouble
- Character.isUpperCase