Supplement to

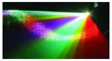
The Design and Implementation of Multimedia Software

The Iterator Pattern

Prof. David Bernstein

James Madison University

users.cs.jmu.edu/bernstdh



Motivation

- Computers vs. Calculators:

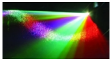
  Computers can perform the same operation many times

- Programming Languages:

  Harness this power using loops

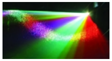
- A Problem:

  Traditional looping requires an understanding of the structure of the "aggregate"



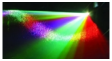
Looping Over an Array

```
String    city;

for (int i=0; i < cities.length; i++)
{
   city = (String)cities[i];
   System.out.println(city);
}
```



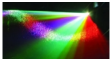
Looping Over an ArrayList

```
String    city;

for (int i=0; i < cities.size(); i++)
{
   city = (String)cities.get(i);
   System.out.println(city);
}
```



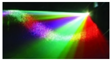
Looping Over a Linked Structure

```
Node      current;
String    city;

current = first;
while (current != null)
{
   city = (String)current.value;
   System.out.println(city);
   current = current.next;
}
```



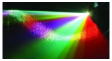
Some Observations

- Applications often “loop” over the same aggregate object in many classes and methods
- Changing from one aggregate to another is, as a result, very inconvenient
- The Iterator design pattern enables us to access the elements of an aggregate object while hiding its internal structure

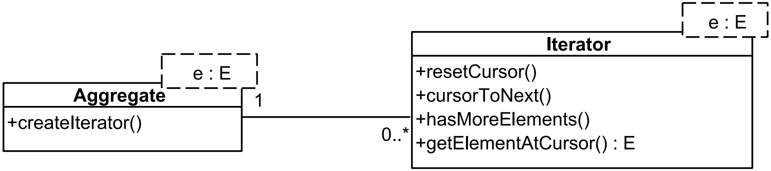


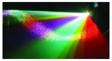
Important Operations

- Reset its “pointer” (or cursor) to the first element
- Determine if there are any more elements in the sequence
- Move its “pointer” to the next element
- Retrieve the “current” element



The Iterator Pattern





Uses in Java

- "Old" Aggregates:

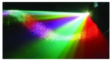
  Aggregates: `Vector`, `Hashtable`

  Iterator: `Enumeration`

- "New" Aggregates:

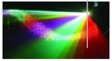
  Aggregates: `ArrayList`, `HashSet`

  Iterator: `Iterator`



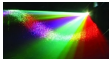
Example - Managing Names with a `Vector`

```
import java.io.*;
import java.util.*;

public class NameList
{
    private Vector<String>   names;


    public NameList()
    {
       names = new Vector<String>();
    }



    public Enumeration<String> elements()
    {
       return names.elements();
    }


    public void read(String fn)
    {
       BufferedReader      in;
       String              line;

       try
```



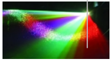
Example - Managing Names with a `Vector` (cont.)

```
        {
           in = new BufferedReader(
              new FileReader(fn));

           while ( (line = in.readLine()) != null)
           {
              names.add(line);
           }

           in.close();
        }
        catch (IOException ioe)
        {
           System.err.println("Problem opening file: "+fn);
           System.exit(1);
        }
     }

}
```



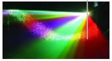
Example - Managing Names with a `Hashtable`

```
import java.io.*;
import java.util.*;

public class NameDatabase
{
    private Hashtable<String, String>     names;


    public NameDatabase()
    {
       names = new Hashtable<String, String>();
    }


    public void add(String name)
    {
       names.put(name, name);
    }



    public Enumeration<String> elements()
    {
       return names.elements();
    }
```



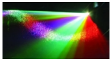
Example - Managing Names with a `Hashtable` (cont.)

```
public void read(String fn)
{
   BufferedReader      in;
   String              line;

   try
   {
      in = new BufferedReader(
         new FileReader(fn));

      while ( (line = in.readLine()) != null)
      {
         add(line);
      }
      in.close();
   }
   catch (IOException ioe)
   {
      System.err.println("Problem opening file: "+fn);
      System.exit(1);
   }
}


public void remove(String name)
{
   names.remove(name);
}
```



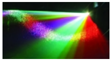
Example - Managing Names with a `Hashtable` (cont.)

```
}
```



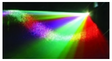
Example - Using the NameList

```
        // To use a NameList
        //
        NameList       names;

        names = new NameList();
```



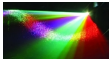
Example - Using the NameDatabase

```
// To use a NameDatabase
//
NameDatabase   names;

names = new NameDatabase();
```



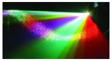
Example - Using Either

```
// Nothing else has to change
//

Enumeration<String>    iterator;
String                 name;


names.read("people.txt");

iterator = names.elements();

while (iterator.hasMoreElements())
{
    name = iterator.nextElement();
    System.out.println(name);
}
```



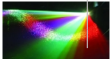
Other Benefits of the Iterator Pattern

- Several objects can be “looping” over the elements in the aggregate at the same time
- A “filtered” list (e.g., names starting with the letter “A”) is handled in eactly the same way that the “unfiltered” version is



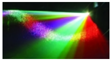
Example - One Iterator used by Many Objects

```
import java.util.*;

public class NamePrinter implements Runnable
{
    private static int   instances = 0;

    private Enumeration  iterator;
    private int          id;
    private Thread       controlThread;



    public NamePrinter(Enumeration names)
    {
       iterator = names;
       instances++;
       id = instances;

       controlThread = new Thread(this);
       controlThread.start();
    }


    public void run()
    {
       int    delay;
       Random random;
       String name;
```



Example - One Iterator used by Many Objects (cont.)

```
       random = new Random(id*System.currentTimeMillis());

       while (iterator.hasMoreElements())
       {
          name = (String)iterator.nextElement();
          System.out.println(id+": "+name);

          try
          {
             delay = random.nextInt(100);
             controlThread.sleep(delay);
          }
          catch (InterruptedException ie)
          {
             // Ignore
          }
       }
    }

}
```



Example - One Iterator used by Many Objects (cont.)

```
import java.util.*;

public class Driver2
{

    public static void main(String[] args)
    {
        Enumeration  iterator;
        NameList     names;
        NamePrinter  np1, np2, np3;

        names = new NameList();
        names.read("people.txt");
        iterator = names.elements();

        np1 = new NamePrinter(iterator);
        np2 = new NamePrinter(iterator);
        np3 = new NamePrinter(iterator);

    }

}
```