CS 213 – Software Methodology

Sesh Venugopal

Design Patterns – 3
Iterator and Template Method Patterns

Iterator Design Pattern

Iterator Design Pattern: Behavioral

```
public class LinkedList<T> {
   public static class Node<E> {
      public E data;
      public Node<E> next;
   }
   public Node<T> front;
   . . .
}
```

Solution 1: Iterate by directly accessing nodes

```
LinkedList<String> list =
                     new LinkedList<String>();
. . .
for (LinkedList.Node<String> ptr = list.front;
    ptr != null; ptr = ptr.next) {
    System.out.println(ptr.data);
}
```

Only works if `front` is accessible to clients, which means it must be made public. Which is simply not acceptable!

Need something like this instead

```
public class LinkedList<T> {
   public static class Node<E> {
      public E data;
      public Node<E> next;
   }
   private Node<T> front;
   . . .
}
```

Iterator: Behavioral

Solution 2: Iterate via method invocation

```
public class LinkedList<T> {
   . . .
   private Node<T> curr;

   public void reset() {
      curr = front;
   }

   public T next() {
      T ret=null;
      if (curr != null) {
         ret = curr.data;
         curr = curr.next;
      }
      return ret;
   }

   public boolean hasNext() {
      return curr != null;
   }

}
```

Basic Iteration using solution 2

```
LinkedList<String> list = new LinkedList<>();
. . .
for (list.reset(); list.hasNext();) {
    System.out.println(list.next());
}
```

E.g. Print #links from each web page to all other web pages

```
LinkedList<URL> list = new LinkedList<URL>();
// populate with web pages . . .
for (list.reset(); list.hasNext();) {
   URL wp1 = list.next();
   for (list.reset(); list.hasNext();) {
      URL wp2 = list.next();
      int n = numLinks(wp1, wp2);
      System.out.println("#links from " + wp1 +
                         " to " + wp2 + " = " + n);
   }
}
```

Will this work?

This won't work – the inner loop thrashes the state of the outer!

Iterator: Behavioral

Solution 3: Separate the Iterator from the LinkedList

```
// in same package as LinkedList
public class LinkedListIterator<T> {

   protected LinkedList.Node<T> curr;

   public LinkedListIterator(
                 LinkedList<T> list) {
      curr = list.front;
   }

   public T next() {
      T ret = null
      if (curr != null) {
          ret = curr.data;
          curr = curr.next;
      }
      return ret;
   }

   public boolean hasNext() {
      return curr != null;
   }
}
```

Print #links from each web page to all other web pages

```
LinkedList<URL> list =
            new LinkedList<URL>();

// populate with web pages . . .

LinkedListIterator<URL> iter1 =
    new LinkedListIterator<URL>(list);

while (iter1.hasNext()) {
   URL wp1 = iter1.next();
   LinkedListIterator<URL> iter2 =
     new LinkedListIterator<URL>(list);
   while (iter2.hasNext()) {
        URL wp2 = iter2.next();
        int n = numLinks(wp1, wp2);
        . . .
    }
}
```

Iterator: Behavioral

Solution 4: Generalization with Interface

Step 1: Have the `LinkedListIterator` class implement an interface

java.util

```
public interface Iterator<T> {
   boolean hasNext();
   T next();
   default void remove() {...}
}
```

```
class LinkedListIterator<T>
                implements Iterator<T> {
   protected LinkedList<T> list;
   protected LinkedList.Node<T> curr;

   LinkedListIterator(LinkedList<T> list) {
      this.list = list;
      curr = list.front;
   }
   public T next() {
      T ret = null
      if (curr != null) {
          ret = curr.data;
          curr = curr.next;
      }
      return ret;
   }
   public boolean hasNext() {
      return curr != null;
   }
   public void remove() {
      throw new
        UnsupportedOperationException();
   }
}
```

By default, removing an item during iteration is not permitted.
This is the implementation of the default remove.
Override if you will allow remove

Iterator: Behavioral

Solution 4: Generalization with Interface

Step 2: Finish up by having the `LinkedList` class implement a method that will return an instance of the `LinkedListIterator`

```
public class LinkedList<T> {
   . . .
   public Iterator<T> iterator() {
      return new
      LinkedListIterator<T>(this);
   }
   . . .
}
```

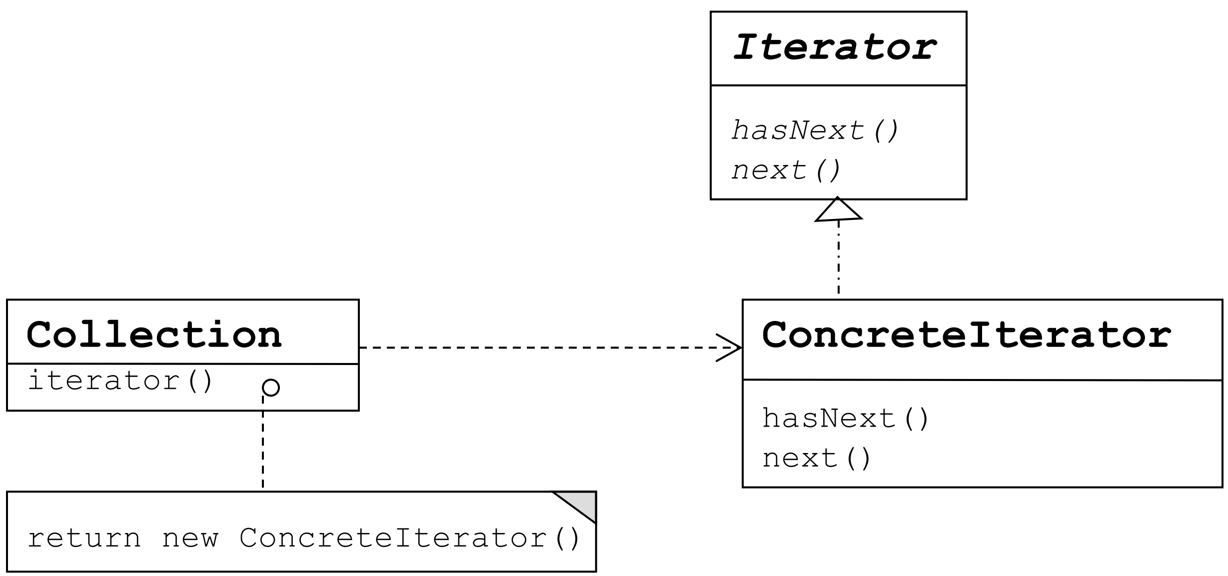
```
LinkedList<URL> list =
     new LinkedList<URL>();
// populate with web pages . . .

Iterator<URL> iter1 = list.iterator();

while (iter1.hasNext()) {
   URL wp1 = iter1.next();
   Iterator<URL> iter2 = list.iterator();
   while (iter2.hasNext()) {
        URL wp2 = iter2.next();
        int n = numLinks(wp1, wp2);
        . . .
    }
}
```

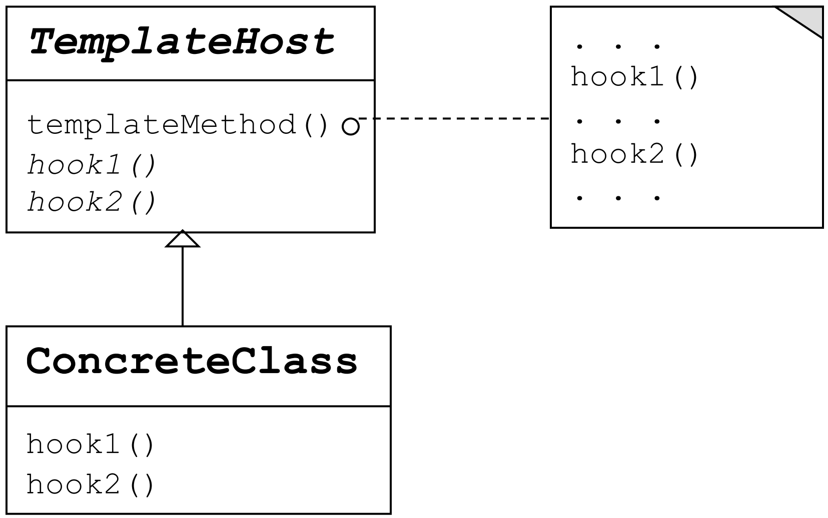
Iterator: Behavioral

- Access the contents of a collection without exposing its internal representation (e.g. hide Node and front)
- Support overlapping multiple traversals (e.g. separate iterator from collection)
- Provide a uniform interface for traversing different collections – support polymorphic iteration (e.g. use interface)



Template Method Design Pattern

Template Method: Behavioral



- A template method implements an algorithm, or a set sequence of actions: some of the actions are abstract because their implementations are specific to concrete subclasses
- The abstract methods are referred to as “hook” methods
- The template method is hosted in an abstract class: note that the template method itself is *not abstract*.
- Each specific algorithm can then extend this abstract host class, and provide its own specific version of the hook method

Example 1: Processing Data

```
public abstract class DataProcessor {
   . . .
   // template method
   public final void process(Resource resource) {
      try {
        open(resource);
        Data data = read(resource);
        processData(data);
        close(resource);
      } catch (OpenCloseException o) {
        reportError(o);
      } catch (ReadException r) {
        reportError(r);
      }
   }

   // non abstract method
   protected void processData(Data data) { ... }

   // hook methods
   protected abstract void open(Resource resource);
   protected abstract Data read(Resource resource);
   protected abstract void close(Resource resource);
   protected abstract void reportError(Exception e);
   . . .
}
```

Example 1: Multiple resource types

```
public class DatabaseProcessor extends DataProcessor {
   . . .
   // implement hook methods
   protected void open(Resource resource) { ... }  // database connection
   protected Data read(Resource resource) { ... }  // SQL statement(s)
   protected void close(Resource resource) { ... } // database connection
   protected void reportError(Exception e) { ... } // write to database log
   . . .
}
public class FileProcessor extends DataProcessor {
   . . .
   // implement hook methods
   protected void open(Resource resource) { ... }  // open file
   protected Data read(Resource resource) { ... }  // read file
   protected void close(Resource resource) { ... } // close file
   protected void reportError(Exception e) { ... } // write to log file
   . . .
}
public class NetworkProcessor extends DataProcessor {
   . . .
   // implement hook methods
   protected void open(Resource resource) { ... }  // open network stream
   protected Data read(Resource resource) { ... }  // read from stream
   protected void close(Resource resource) { ... } // close network stream
   protected void reportError(Exception e) { ... } // write to a network location
   . . .
}
```

Example 1: Application Calls

```
// use database
DataProcessor dproc = new DatabaseProcessor();
Resource dresource = new DatabaseResource();
. . .
dproc.process(dresource);
```

```
// use file
DataProcessor dproc = new FileProcessor();
Resource dresource = new FileResource();
. . .
dproc.process(dresource);
```

```
// use network
DataProcessor dproc = new NetworkProcessor();
Resource dresource = new NetworkResource();
. . .
dproc.process(dresource);
```

Example 2 – Graph DFS

Since depth-first search serves as a basis for various graph algorithms, it can be implemented with template methods that can then be overridden appropriately by DFS-based algorithms/applications

Key observation: The base DFS code does the traversal through the graph, while providing hooks for:

- Restarting DFS at different vertices
- Doing stuff on getting to a vertex
- Doing stuff just before leaving a vertex (to back up to previous recursive level)

Example 2 – Graph DFS

```
public abstract class DFS {
   protected Graph G;
   protected boolean[] visited;
   protected int[] info;

   public DFS(Graph G) {
       this.G = G; visited = new boolean[G.n];
       for (int v=0; v < G.n; v++) {
           visited[v] = false;
       }
       info = new int[G.n];
   }

   public final int[] dfs() {  // template method
     ...
   }

   protected final void dfs(int v) {  // template method
     ...
   }

   ...
}
```

Example 2 – Graph DFS

```
public abstract class DFS
   ...

   public final int[] dfs() {  // template method
      for (int v=0; v < G.n; v++) {
          if (!visited[v]) {
             restart();
             dfs(v);
          }
      }
      return info;
   }

   protected final void dfs(int v) {  // template method
      preAction(v); visited[v] = true;
      Iterator<Integer> iter = G.neighborsIterator(v);
      while (iter.hasNext()) {
         int v = iter.next();
         if (!visited[v]) { dfs(v); }
      }
      postAction(v);
   }

   protected abstract void restart();          // hook 1
   protected abstract void preAction(int v);   // hook 2
   protected abstract void postAction(int v);   // hook 3
}
```

Example 2: Topological Sort

```
public class Topsort extends DFS {

   protected int topNum;

   public Topsort(Graph G) {
      super(G);
      topNum = n-1;
   }

   // hook methods, redefined
   protected void restart() { }          // do nothing
   protected void preAction(int v) { }   // do nothing

   protected void postAction(int v) {    // slot v in sequence
      info[topNum--] = v;
   }
}
```

USAGE:

```
DFS topsort = new Topsort(graph);
int[] topSequence = topsort.dfs();
```

Example 2: Connected Components

```
public class ConnComp extends DFS {

   protected int currComp;

   public Conncomp(Graph G) {
      super(G);
      currComp = 0;
   }

   // hook methods, redefined
   protected void restart() { currComp++; } // for next component
   protected void preAction(int v) { info[v] = currComp; }

   protected void postAction(int v) { }  // do nothing
}
```

USAGE:

```
DFS connectedComps = new ConnComp(graph);
int[] components = connectedComps.dfs();
```