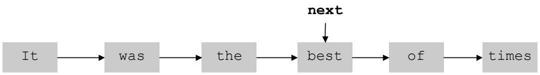
2.7 Lists and Iterators

Sequences and Urns

Sequence. Ordered collection of items.

Key operations. Insert an item, iterate over the items.

Design challenge. Support iteration by client, without revealing the internal representation of the collection.



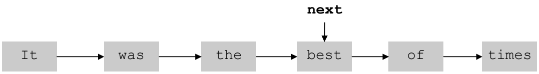
Iteration in Java

Iterator Interface

API for `java.util.Iterator`.

- `hasNext()` Are there more items in the list?
- `next()` Return the next item in the list.
- `remove()` Delete the last item returned by `next()`.

```
public interface Iterator<Item> {
   boolean hasNext();
   Item next();
   void remove();   // optional
}
```



Iterator Client

API for **java.util.Iterator**.

- `hasNext()` Are there more items in the list?
- `next()` Return the next item in the list.
- `remove()` Delete the last item returned by `next()`.

```
public static void main(String[] args) {
   Sequence<String> list = new Sequence<String>();
   list.add("This");
   list.add("is");
   list.add("a");
   list.add("test.");
   Iterator<String> i = list.iterator();
   while (i.hasNext()) {
      String s = i.next();
      System.out.println(s);
   }
}
```

Iterable Interface

API for **java.lang.Iterable**.

- `iterator()` Return an iterator.

```
public interface Iterable<Item> {
   Iterator<Item> iterator();
}
```

Ex. `Sequence`, `java.util.ArrayList`, `HashSet`.

Enhanced For Loop

Enhanced for loop. Syntactic sugar for iterating over a collection.

```
public static void main(String[] args) {
   Sequence<String> list = new Sequence<String>();
   list.add("This");
   list.add("is");
   list.add("a");
   list.add("test.");
   for (String s : list)
      System.out.println(s);
}
```

implements `Iterable` ⇒
can iterate using enhanced for loop

Remark. Can also use enhanced for loop with arrays.

Sequence ADT: Two Implementations

Sequence: Linked List Implementation

```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class Sequence<Item> implements Iterable<Item> {
   private Node first, last;

   private class Node { Item item; Node mext; }

   public void add(Item item) {
      Node x = new Node();
      x.item = item;
      if (first == null) first = x;
      else last.next = x;
      last = x;
   }
```

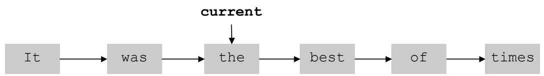
same as queue

```
   public Iterator<Item> iterator() {
      return new SeqIterator();
   }
```

next slide

Sequence: Linked List Implementation (cont)

```
private class SeqIterator implements Iterator<Item> {
   Node current = first;

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

   public void remove() {
      throw new UnsupportedOperationException();
   }

   public Item next() {
      if (!hasNext()) throw new NoSuchElementException();
      Item item = current.item;
      current = current.next;
      return item;
   }
}
```



Sequence: Array Implementation

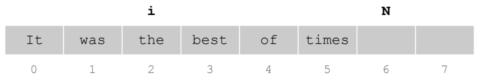
```
import java.util.Iterator;
import java.util.NoSuchElementException;

public class Sequence<Item> implements Iterable<Item> {
   private Item[] a = (Item[]) new Object[8];
   private int N = 0;

   public void add(Item item) {
      if (N >= a.length) resize();
      a[N++] = item;
   }
```

as usual, with array doubling

```
   public Iterator<Item> iterator() {
      return new SeqIterator();
   }

   private class SeqIterator
      // see next slide
}
```

Sequence: Array Implementation (cont)

```
private class SeqIterator implements Iterator<Item> {
   int i = 0;

   public boolean hasNext() { return i < N; }

   public void remove() {
      throw new UnsupportedOperationException();
   }

   public Item next() {
      if (!hasNext()) throw new NoSuchElementException();
      return a[i++];
   }
}
```



Applications

Load Balancing

Load balancing. N users want to choose among N identical file shares. The goal is to balance users across file shares. Assume it's too hard to coordinate (or query) all resources to see how empty they are.

Random assignment. Assign each user to a resource at random.

```
% java LoadBalance 10
0:
1:
2: user7
3: user1 user2 user8
4: user0 user9
5:
6: user3 user6
7:
8: user4
9: user5
                         max load = 3
```

Server.java

```
public class Server {
   private Sequence<String> list = new Sequence<String>();
   private int load;

   public void add(String user) {
      list.add(user);
      load++;
   }

   public String toString() {
      String s = "";
      for (String user : list)
         s += user + " ";
      return s;
   }
}
```

Load Balancing

```
public class LoadBalance {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      Server[] servers = new Server[N];
      for (int i = 0; i < N; i++)
         servers[i] = new Server();

      // assign N users to N servers at random
      for (int j = 0; j < N; j++) {
         String user = "user" + j;
         int i = (int) (Math.random() * N);
         servers[i].add(user);
      }

      // print results
      for (int i = 0; i < N; i++)
         System.out.println(i + ": " + servers[i]);
   }
}
```

Load Balancing

Load balancing. N users want to choose among N identical file shares. The goal is to balance users across file shares. Assume it's too hard to coordinate (or query) all resources to see how empty they are.

Coordinated assignment. Assign user i to server i.
Result. Max load = 1.

Random assignment. Assign each user to a resource at random.
Theory. Max load ≈ log N / log log N.

Best of two. For each user, choose two resources at random and assign user to least busy one.
Theory. Max load ≈ log log N.

Java List Libraries: ArrayList and LinkedList

API for `java.util.ArrayList`.

- `add()` Add item to end of list.
- `iterator()` Return an iterator to the list.
- `size()`, `remove()`, `set()`, `clear()`, `indexOf()`, `toArray()`, ….

```
import java.util.ArrayList;

public class Test {
   public static void main(String[] args) {
      ArrayList<String> list = new ArrayList<String>();
      list.add("This");
      list.add("is");
      list.add("a");
      list.add("test.");
      for (String s : list)
         System.out.println(s);
   }
}
```

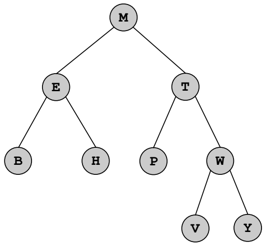
Tree Iterators

Binary Tree Iterator

Binary tree. Create an iterator for a binary tree.
(and avoid using extra space)

```
public class BinaryTree<Item> {
   private Node root;

   private class Node {
      Item item;
      Node l, r;
   }

   public Iterator<Item> iterator() {
      return new Preorder();
   }
}
```

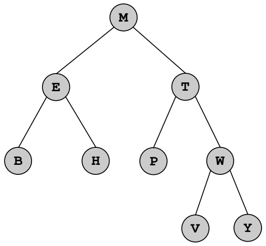
goal: implement this



Preorder Traversal

Preorder traversal. Visit a node before its two children.

```
private static void preorder(Node x) {
   if (x == null) return;
   System.out.println(x.item);
   preorder(x.l);
   preorder(x.r);
}
```



preorder: M E B H T P W V Y

Q. How to implement an iterator for preorder traversal?

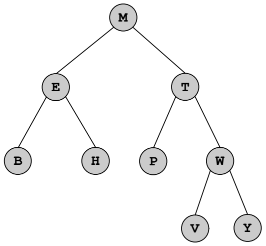
Binary Tree Iterator: Preorder Traversal

```
private class Preorder implements Iterator<Item> {
   Stack<Node> stack = new Stack<Node>();

   Preorder() {
      if (root != null) stack.push(root);
   }

   public void remove() { // throw exception as before }

   public boolean hasNext() { return !stack.isEmpty(); }

   public Item next() {
      if (!hasNext()) throw new NoSuchElementException();
      Node x = stack.pop();
      Item item = x.item;
      if (x.r != null) stack.push(x.r);
      if (x.l != null) stack.push(x.l);
      return item;
   }
}
```

Binary Tree Iterator: Level Order

Level order. Examine nodes in order of distance from root.

Q. How to implement an iterator for level order traversal?



level order: M E T B H P W V Y