Inner Classes

- In Java (and C++), a class can be nested within another class.
- Each object in the inner class exists relative to an object of the outer class.
- Objects of the inner class have available instance variables and methods of the outer class.

Ways to Construct ClosedList

- `class Cell {...}
class ClosedList {...}` - Separate Classes
- `class ClosedList
  
  class Cell {...}

  ...` - Inner Class

Interpretation of Identifiers

- In an inner class, the innermost meaning of an identifier applies.

```java
class ClosedList
{
  String identity;
  class Cell
  {
    String identity;
    ...
  }
  ...
}
```

Usage

- Normally one or more objects of the inner class are created for a given object of the outer class.
- Objects of the inner class only make sense in the context of a supporting object of the outer class.

Exporting Inner Objects

- Inner objects can be used outside, understanding that they are always relative to the object in which they were created.
Example: List Iterator

- We want to define an Iterator for a ClosedList.
- For read-only Iteration, the Iterator class can be defined outside the ClosedList class.
- For modification, such as remove(), it is sometimes necessary for the Iterator to change variables in the container, such as the head or tail.

ClosedList.Iterator

class ClosedList
{
    private Cell head;
    private Cell tail;

    public static void main(String arg[])
    {
        int numItems = 10;
        ClosedList L = new ClosedList();
        for( int i = 0; i < numItems; i++ )
        {
            L.enqueue(new Integer(i));
        }
        ClosedList.Iterator it = L.getIterator();
        System.out.println("removing " + it.next());
        it.remove(); // remove first item
    }
}

Example: List Iterator (2)

- By making the ListIterator an inner class, we can:
  - Use data elements defined in the ClosedList.
  - Avoid exposing those data elements to the world at large.
  - Use Iterators outside ClosedList.

ClosedList.Iterator: remove()

Defined to remove the value just produced by next().

```java
public void remove()
{
    if (previous == null)
    {
        head = head.getNext();
    }
    else
    {
        previous.setNext(current.getNext()); // reuse previous
        current = previous; // lose current
    }
}
```

Test Program

class TestClosedList
{
    public static void main(String arg[])
    {
        int numItems = 10;
        ClosedList L = new ClosedList();
        // add 10 items to L
        for( int i = 0; i < numItems; i++ )
        {
            System.out.println("initial list contents: " + L);
            L.enqueue(new Integer(i));
        }
        System.out.println("List contents after adding items: " + L);
        // remove items from the beginning. list 9 items
        ClosedList.Iterator it = L.getIterator();
        it.remove(); // remove first item
        System.out.println("List contents after removing item: " + L);
        System.out.println("removing " + it.next());
        it.remove(); // remove first item
        System.out.println("List contents after removing item: " + L);
    }
}
## Test Program Output

<table>
<thead>
<tr>
<th>Initial list contents: 0 1 2 3 4 5 6 7 8 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>skipping 0</td>
</tr>
<tr>
<td>skipping 1</td>
</tr>
<tr>
<td>skipping 2</td>
</tr>
<tr>
<td>removing 3</td>
</tr>
<tr>
<td>removing 4</td>
</tr>
<tr>
<td>List contents after removing two: 0 1 2 5 6 7 8 9</td>
</tr>
<tr>
<td>skipping 5</td>
</tr>
<tr>
<td>skipping 6</td>
</tr>
<tr>
<td>skipping 7</td>
</tr>
<tr>
<td>inserting 10</td>
</tr>
<tr>
<td>inserting 20</td>
</tr>
<tr>
<td>inserting 30</td>
</tr>
<tr>
<td>List contents after inserting three: 0 1 2 5 6 7 10 20 30 8 9</td>
</tr>
</tbody>
</table>

```