
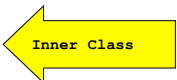


Inner Classes

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.

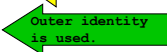
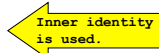
Ways to Construct ClosedList

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

Interpretation of Identifiers

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

```
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.

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

```
class ClosedList
{
    private Cell head;
    private Cell tail;
    ...
    public Iterator getIterator()
    {
        return new Iterator(head);
    }

    public class Iterator // inner class to ClosedList
    {
        private Cell current;
        private Cell previous; // keep track of previous

        public Iterator(Cell head)
        {
            current = head;
            previous = null;
        }
        ...
    }
}
```

Can export Iterator to outside!

ClosedList.Iterator: remove()

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

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

head is defined in the outer class!

Test Program

```
class TestClosedList
{
    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
    }
}
```

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++ )
        {
            L.enqueue(new Integer(i));
        }

        System.out.println("Initial list contents: " + L);

        // starting from the beginning, skip 3 items
        ClosedList.Iterator it = L.getIterator();
        for( int i = 0; i < 3; i++ )
        {
            System.out.println("skipping " + it.next());
        }

        // remove 2 items
        System.out.println("removing " + it.next());
        it.remove();
        System.out.println("removing " + it.next());
        it.remove();

        System.out.println("List contents after removing two: " + L);

        for( int i = 0; i < 3; i++ )
        {
            System.out.println("skipping " + it.next()); // ignore value
        }

        // insert 3 items
        for( int i = 0; i < 3; i++ )
        {
            int value = 10*(i+1);
            System.out.println("inserting " + value);
            it.insert(new Integer(value));
        }

        System.out.println("List contents after inserting three: " + L);
    }
}
```

Test Program Output

```
Initial list contents: 0 1 2 3 4 5 6 7 8 9
skipping 0
skipping 1
skipping 2
removing 3
removing 4
List contents after removing two: 0 1 2 5 6 7 8 9
skipping 5
skipping 6
skipping 7
inserting 10
inserting 20
inserting 30
List contents after inserting three: 0 1 2 5 6 7 10 20 30 8 9
```