Linked Lists and Testing

CSE 373
Data Structures
Winter 2006

Agenda

• A new implementation of lists using single-linked list data structures (review)
• Testing
  › Goals
  › Unit testing
  › Automated testing with JUnit

Last Time

• Interfaces: BasicList, BasicListIterator
  › Specifies list operations essentially the same as ones in Java collection classes
• Implementation: BasicArrayList
  › A particular implementation using an array as the backing store
  › Dynamically expanding array – appears “infinite” to clients

Today’s Example

• Same interfaces: BasicList, BasicListIterator
• Implementation: BasicLinkedList
  › Implemented with a single-linked list as the backing store
  › Also appears “infinite” to clients
  (Note: initial version is very simplistic – we’ll improve on it over the next lecture or two)

BasicLinkedList Nodes

• Each link in the list is an instance of the following (local) class
  private class Link {
    public Object item;    // list element referenced
    // by this link
    public Link next;      // next link or null if this is the last
    // link in the list
    // constructor for convenience
    public Link(Object item, Link next) { … }
  }

List Representation

• We can implement a BasicLinkedList with (only) the following instance variable
  private Link head;    // reference to first link in
  // the list, or null if the
  // list is empty
  (Of course, additional instance data may make it easier to do some things faster, but this is enough to get started.)
Typical List Operation

```java
public int indexOf(Object obj) {
    // sequential search
    int pos = 0;       // position of current link in the list
    Link p = head;
    while (p != null) {
        if (p.item.equals(obj)) {
            return pos;
        }
        p = p.next;
        pos++;
    }
    return -1;
}
```

Another List Operation

```java
public int size() {
    // count the number of links in the list
    int nItems = 0;
    Link p = head;
    while (p != null) {
        nItems++;
        p = p.next;
    }
    return nItems;
}
```

But wait!! This takes O(n) time!!! We should be able to do better – and we can

Speeding up size()

• Instead of counting the links, keep the list length in a separate instance variable, updated as needed
• A typical example of trading storage for computation
• But how do we verify that we don’t break anything if we make this change?
  › And how do we know that things are ok to start with?

Testing & Debugging

• Testing
  › Verify that things work as expected
  › Be able to reverify as software evolves
• Debugging
  › Controlled experiment to discover what is wrong and where

Testing Strategies

• Test "typical" cases – basic functional tests
  › Do operations work properly on a non-empty list?
• Test "edge" cases
  › Zero, one, many (empty list, single element, more, …)
  › Limit cases – what happens if a container is full
  › Error cases – do things blow up as expected (index out of bounds, other exceptions)
• Stress tests – hard, but needed in production code – what happens under large workloads

Debugging Strategies

• Questions to ask
  › What’s wrong?
  › What’s working? How far do we get before something fails?
  › What are the symptoms?
  › What changed since the last time it worked?
• Observing strategies
  › Print statements(!)
  › Debuggers – CAT scans for software
  › Etc…
Unit Tests

- Idea: first set of tests: a collection of tests for individual operations
- Effective testing: lots of small tests, each of which checks something specific
  › (Avoid “big-bang” tests as your only strategy)

Where to Put Tests

- Type them in using the programming environment (tedious)
- Lots of test programs (better – don’t have to retype – but still tedious to run repeatedly)
- Automated test frameworks
  › Been around for a while, but popularized by “extreme programming” / “agile development” movements in recent years

JUnit

- Test framework for Java unit tests
- Implemented as classes that extend Junit’s TestCase class
- Key: test methods are named testXXXX
- Optional: setUp() method to create state before each individual test is run
- More, but these are the core ideas

Inside Test Methods

- Inherited from TestCase; typical ones include
  assertEquals(expected, actual)
  assertEquals(expected, actual, delta) // doubles
  assertTrue(condition)
  assertFalse(condition)
  assertNull(object)
  assertNotNull(object)
  Fail("message") // generate failure if control
  // should not reach a particular point

Unit Test Strategy

- Define tests before or as you write code
- Add and run tests each time you add something small to the code
- Rerun tests to verify nothing broken after changes
- If a bug is detected, create a test to demonstrate it, fix it, then keep the test forever as part of the test suite