CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2002
Doc 4 Binary Tree Example
Contents

Exercise

This document contains some uncommented code. While you are reading this document keep a list of places were you wish there were comments. Which comments would you have liked to see in the code?

Binary Tree Example

Here is an example of object recursion and Null Object in a binary tree.

First the Java version then the Smalltalk version

Java Version
Node

```
public interface Node
  {
  public Object put( int key, Object value);
  public Object get( int key);
  public boolean containsKey(int key );
  }
```

BinaryNode

```
public class BinaryNode implements Node
   {
   private int key;
   private Object value;
   private Node left;
   private Node right;

   public BinaryNode(int key, Object value )
      {
      this.key = key;
      this.value = value;
      right = left = new LeafNode(this);
      }

   Object privatePut( int newKey, Object newValue)
      {
      if (newKey > key)
         right = new BinaryNode(newKey, newValue);
      else
         left = new BinaryNode(newKey, newValue);
      return null;
      }

   public Object put( int newKey, Object newValue)
      {
      if (this.key == newKey)
         {
         Object oldValue = value;
         value = newValue;
         return oldValue;
         }
      if (newKey > key)
         return right.put(newKey, newValue);
      else
         return left.put(newKey, newValue);
      }
```

BinaryNode Continued

```
public Object get( int findKey)
   {
   if (this.key == findKey)
      return value;
   if (findKey > key)
      return right.get(findKey);
   else
      return left.get(findKey);
   }

public boolean containsKey(int findKey )
   {
   if (this.key == findKey)
      return true;
   if (findKey > key)
      return right.containsKey(findKey );
   else
      return left.containsKey(findKey);
   }

public String toString()
   {
   return "(" + left + key + right + ")";
   }

}
```

LeafNode

```
public class LeafNode implements Node
  {
  private BinaryNode parent;

  public LeafNode(BinaryNode parent )
    {
    this.parent = parent;
    }

  public Object put( int key, Object value)
    {
    return parent.privatePut(key, value);
    }

  public Object get( int key)
    {
    return null;
    }

  public boolean containsKey(int key )
    {
    return false;
    }

  public String toString()
    {
    return "";
    }
  }
```

TestCases

```
import junit.framework.TestCase;

public class TestNodes extends TestCase
   {
   public TestNodes (String name )
      {
      super(name);
      }

   public void testPut()
      {
      Node root = new BinaryNode(10, "10");
      root.put(5, "5");
      root.put(3, "3");
      root.put(20, "20");
      root.put(6, "6");
      root.put(1, "1");
      root.put(4, "4");
      assert("10", root.containsKey( 10 ));
      assert("5", root.containsKey( 5 ));
      assert("3", root.containsKey( 3 ));
      assert("20", root.containsKey( 20 ));
      assert("6", root.containsKey( 6 ));
      assert("1", root.containsKey( 1 ));
      assert("4", root.containsKey( 4 ));
      assertEquals("Find -1", root.containsKey( -1), false);
      }

   public void testToString()
      {
      Node root = new BinaryNode(10, "10");
      root.put(5, "5");
      root.put(3, "3");
      root.put(20, "20");
      root.put(6, "6");
      root.put(1, "1");
      root.put(4, "4");
      assertEquals("tostring", root.toString(), "((((1)3(4))5(6))10(20))");
      }
```

```
public void testGet()
    {
    Node root = new BinaryNode(10, "10");
    root.put(5, "5");
    root.put(3, "3");
    root.put(20, "20");
    root.put(6, "6");
    root.put(1, "1");
    root.put(4, "4");
    assertEquals("10", root.get( 10 ), "10");
    assertEquals("5", root.get( 5 ), "5");
    assertEquals("3", root.get( 3 ), "3");
    assertEquals("20", root.get( 20 ), "20");
    assertEquals("6", root.get( 6 ), "6");
    assertEquals("1", root.get( 1 ), "1");
    assertEquals("4", root.get( 4 ), "4");
    assertEquals("-1", root.get( -1), null);
    }
}
```

Smalltalk Version
BinaryNode

```
Smalltalk.CS635 defineClass: #BinaryNode
    superclass: #{Core.Object}
    indexedType: #none
    private: false
    instanceVariableNames: 'key value left right '
    classInstanceVariableNames: ''
    imports: ''
    category: 'Assignment-1&2'
```

CS635.BinaryNode class methods

```
key: aKey value: anObject
    ^super new
        setKey: aKey
        value: anObject
```

CS635.BinaryNode Instance methods

```
setKey: aKey value: anObject
    key := aKey.
    value := anObject.
    right := left := LeafNode parent: self.

at: aKey
    aKey = key ifTrue:[^value].
    aKey < key ifTrue: [^left at: aKey].
    aKey > key ifTrue: [^right at: aKey].

at: aKey put: anObject
    | oldValue |
    aKey < key ifTrue: [^left at: aKey put: anObject].
    aKey > key ifTrue: [^right at: aKey put: anObject].
    oldValue := value.
    value := anObject.
    ^oldValue
```

```
privateAt: aKey put: anObject
    aKey < key ifTrue: [left := BinaryNode key: aKey value: anObject].
    aKey > key ifTrue: [right := BinaryNode key: aKey value: anObject].
    ^anObject

printOn: aStream
    aStream
        nextPutAll: '(';
        print: left;
        print: key;
        print: right;
        nextPutAll: ')'

includes: aKey
    aKey = key ifTrue:[^true].
    aKey < key ifTrue:[^left includes: aKey].
    aKey > key ifTrue:[^right includes: aKey].
```

LeafNode

```
Smalltalk.CS635 defineClass: #LeafNode
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: 'parent '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Assignment-1&2'
```

CS635.LeafNode class methods

```
parent: aNode
   ^super new setParent: aNode

setParent: aNode
   parent := aNode

at: aKey
   self keyNotFoundError: aKey

at: aKey put: anObject
   parent privateAt: aKey put: anObject

keyNotFoundError: aKey

   ^Dictionary keyNotFoundSignal raiseWith: aKey

printOn: aStream

includes: aKey
   ^false
```

Tests

```
Smalltalk.CS635 defineClass: #TestBinaryTree
   superclass: #{XProgramming.SUnit.TestCase}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: ''
   imports: ''
   category: 'Assignment-1&2'
```

```
testAdd
   | root |
   root := BinaryNode key: 10 value: 10.
   root
      at: 5 put: 5;
      at: 12 put: 12;
      at: 3 put: 3;
      at: 6 put: 6;
      at: 1 put: 1.
   self
      assert: (root includes: 5);
      assert: (root includes: 12);
      assert: (root includes: 3);
      assert:  (root includes: 6);
      deny: (root includes: -1);
      deny: (root includes: 7);
      deny: (root includes: 200).
   root := BinaryNode key: 10 value: 10.
   self
      assert: (root includes: 10);
      deny: (root includes: 1);
      deny: (root includes: 200).
```

```
testAt
   | root |
   root := BinaryNode key: 10 value: '10'.
   root
      at: 5 put: '5';
      at: 12 put: '12';
      at: 3 put: '3';
      at: 6 put: '6';
      at: 1 put: '1'.
   self
      assert: (root at: 5) = '5';
      assert: (root at: 12) = '12';
      assert: (root at: 3) = '3';
      assert:  (root at: 6) = '6'.
   self
      should: [root at: -1]
      raise: KeyedCollection keyNotFoundSignal.
```

Comments
Recursive Implementation

On skinny trees the recursive nature of this code can cause a stack overflow

Unless you have balanced trees (AVL, Red-Black, etc) an iterative solution would avoid the stack overflow

Space Requirements

A complete binary tree with N internal nodes (BinaryNode) has 2N leafs.

The above implementation uses less than 2N LeafNodes by using the same LeafNode object for the left and right subtree of a BinaryNode object. For large N this still may require too much space.

It is possible to use only one LeafNode in a program. The only state in the LeafNode is the parent pointer. One can remove this state by making the parent pointer an argument in the at:put: & put() methods. This requires a different public and private at:put: & put() methods. This can be handled in a BinaryTree class. Doing this makes these methods a bit odd.

In Smalltalk there is a nice trick to using only one LeafNode. Smalltalk have a pseudo variable thisContext that contains information about the sender of a method. This allows a method to determine which object sent the method.

The following shows the changes needed to use a single instance of LeafNode.

Single LeafNode Example

```
Smalltalk.CS635 defineClass: #LeafNode
   superclass: #{Core.Object}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: 'instance '
   imports: ''
   category: 'Assignment-1&2'
```

CS635.LeafNode class methodsFor: 'instance creation'

```
instance
   instance isNil ifTrue:[instance := self new].
   ^instance
```

CS635.LeafNode methods

```
at: aKey
   self keyNotFoundError: aKey

at: aKey put: anObject
   | callingMethod parentNode |
   callingMethod := thisContext sender.
   parentNode := callingMethod receiver.
   parentNode privateAt: aKey put: anObject

keyNotFoundError: aKey
   ^Dictionary keyNotFoundSignal raiseWith: aKey

printOn: aStream

includes: aKey
   ^false
```

Changes to the BinaryNode

```
setKey: aKey value: anObject
   key := aKey.
   value := anObject.
   right := left := LeafNode instance.
```