Genevieve Leach

CS 241 – 01

1 November 2016

Project 1

**Source Code****import** java.util.Scanner;  
  
**public class** LeachGp1 {  
  
 **public static** Node *root* = **null**;  
  
 **public static void** main(String[] args) {  
 Scanner kb = **new** Scanner(System.***in***);  
 String input = **null**;  
 String menu = **"Enter A for add, D for delete, P for print, Q for quit."**;  
  
 System.***out***.println(**"Welcome to the Binary Search Tree. What would you like**

**to do?"**);  
 **while**(**true**) { *//used to keep repeating until closed with the quit option* System.***out***.println(menu);  
  
 input = kb.nextLine();  
 input = input.trim(); *//removes any leading/trailing spaces* input = input.toUpperCase(); *//can accept lower/uppercase input* **switch** (input) {  
 **case "A"**: {  
 System.***out***.println(**"What number would you like to add?"**);  
 **int** num = kb.nextInt();  
 *add*(num);  
 kb.nextLine(); *//consumes any unused characters in last input* **break**;  
 }  
 **case "D"**: {  
 **if** (*root* == **null**) {  
 System.***out***.println(**"Tree is empty, cannot delete anything**

**more."**);  
 **break**;  
 }  
 System.***out***.println(**"What number would you like to delete?"**);  
 **int** num = kb.nextInt();  
 *delete*(num);  
 kb.nextLine(); *//consumes any unused characters in last input* **break**;  
 }  
 **case "P"**: {  
 *print*();  
 **break**;  
 }  
 **case "Q"**: {  
 System.***out***.println(**"Thank you for using the tree. Good bye."**);  
 System.*exit*(0);  
 }  
 **default**: {  
 System.***out***.println(**"Invalid input. Please try again."**);  
 }  
 }  
 }  
 }  
  
 **public static void** add(**int** a) {  
 **if**(*root* == **null**) {  
 *root* = **new** Node(a);  
 System.***out***.println(**"Value "** + a + **" added."**);  
 } **else if** (*search*(a)) {  
 System.***out***.println(**"Value "** + a + **" is already in tree."**);  
 } **else** {  
 Node toAdd = **new** Node(a);  
 Node cursor = *root*;  
 Node peek = **null**;  
  
 **while**(**true**) {  
 **if**(a < cursor.**data**) {  
 peek = cursor.**leftChild**;  
 **if** (peek == **null**) {  
 cursor.**leftChild** = toAdd;  
 System.***out***.println(**"Value "** + a + **" added."**);  
 **return**;  
 } **else** {  
 cursor = cursor.**leftChild**;  
 }  
 } **else** {  
 peek = cursor.**rightChild**;  
 **if**(peek==**null**) {  
 cursor.**rightChild** = toAdd;  
 System.***out***.println(**"Value "** + a + **" added."**);  
 **return**;  
 } **else** {  
 cursor = cursor.**rightChild**;  
 }  
 }  
 }  
  
 }  
 }  
  
 **public static void** delete(**int** a) {  
 **if** (!*search*(a)) {  
 System.***out***.println(**"Value "** + a + **" is not in the tree."**);  
 } **else if** (*root*.**data** == a) {

*//if it is a leaf*  
 **if** (*root*.**leftChild** == **null** && *root*.**rightChild** == **null**) {  *root* = **null**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 } *//if it only has 1 child*

**else if** (*root*.**leftChild** == **null** && *root*.**rightChild** != **null**) {  *root* = *root*.**rightChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 } *//if it only has 1 child*

**else if** (*root*.**rightChild** == **null** && *root*.**leftChild** != **null**) {  *root* = *root*.**leftChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 } *//if it has 2 children*

**else** {  *twoChildrenDeletion*(*root*, a);  
 **return**;  
 }  
 } **else** {  
 Node cursor = *root*;  
 Node peek = **null**;  
 **while** (**true**) {  
 **if** (a < cursor.**data**) {  
 peek = cursor.**leftChild**;  
 **if** (peek.**data** == a) {

*//if it is a leaf*  
 **if** (peek.**leftChild** == **null** && peek.**rightChild** == **null**) { cursor.**leftChild** = **null**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it only has 1 child*

**else if** (peek.**leftChild** == **null** && peek.**rightChild** != **null**) { cursor.**leftChild** = peek.**rightChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it only has 1 child*

**else if** (peek.**rightChild** == **null** && peek.**leftChild** != **null**) { cursor.**leftChild** = peek.**leftChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it has 2 children*

**else** {  *twoChildrenDeletion*(peek, a);  
 **return**;  
 }  
 } **else** {  
 cursor = cursor.**leftChild**;  
 }  
 } **else if** (a > cursor.**data**) {  
 peek = cursor.**rightChild**;  
 **if** (peek.**data** == a) {

*//if it is a leaf*  
 **if** (peek.**leftChild** == **null** && peek.**rightChild** == **null**) { cursor.**rightChild** = **null**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it only has 1 child*

**else if** (peek.**leftChild** == **null** && peek.**rightChild** != **null**) { cursor.**rightChild** = peek.**rightChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it only has 1 child*

**else if** (peek.**rightChild** == **null** && peek.**leftChild** != **null**) { cursor.**rightChild** = peek.**leftChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//if it has 2 children*

**else** {  *twoChildrenDeletion*(peek, a);  
 **return**;  
 }  
 } **else** {  
 cursor = cursor.**rightChild**;  
 }  
 }  
 }  
 }  
 }  
  
 **public static void** print() {  
 **if**(*root* == **null**) {  
 System.***out***.println(**"Tree is empty."**);  
 } **else** {  
 System.***out***.println(**"Preorder traversal:"**);  
 *preorder*(*root*);  
 System.***out***.println();  
 System.***out***.println(**"Inorder traversal:"**);  
 *inorder*(*root*);  
 System.***out***.println();  
 System.***out***.println(**"Postorder traversal:"**);  
 *postorder*(*root*);  
 System.***out***.println();  
 }  
 }  
  
 *//following methods for traversals use recursion to print* **public static void** preorder(Node cursor) {  
 **if**(cursor == **null**) {  
 **return**;  
 }  
 System.***out***.print(cursor.**data** + **", "**);  
 *preorder*(cursor.**leftChild**);  
 *preorder*(cursor.**rightChild**);  
 }  
  
 **public static void** inorder(Node cursor) {  
 **if** (cursor == **null**) {  
 **return**;  
 }  
 *inorder*(cursor.**leftChild**);  
 System.***out***.print(cursor.**data** + **", "**);  
 *inorder*(cursor.**rightChild**);  
 }  
  
 **public static void** postorder(Node cursor) {  
 **if** (cursor == **null**) {  
 **return**;  
 }  
 *postorder*(cursor.**leftChild**);  
 *postorder*(cursor.**rightChild**);  
 System.***out***.print(cursor.**data** + **", "**);  
 }  
  
 *//to see if a value is already in the tree before adding* **public static boolean** search(**int** a) {  
 Node cursor = *root*;  
 **while** (cursor!=**null**) {  
 **if**(cursor.**data** == a) {  
 **return true**;  
 } **else if** (cursor.**data** > a) {  
 cursor = cursor.**leftChild**;  
 } **else** {  
 cursor = cursor.**rightChild**;  
 }  
 }  
 **return false**;  
 }  
  
 *//node with 2 children deletion* **public static void** twoChildrenDeletion(Node cursor, **int** a) {  
 Node peek = cursor.**rightChild**;  
 Node peekParent = cursor;

*//if the right node is the replacement*  
 **if**(peek.**leftChild** == **null**) { cursor.**data** = peek.**data**;  
 cursor.**rightChild** = peek.**rightChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 **return**;  
 } *//else find the replacement node, replace data in peek* **while**(peek.**leftChild** != **null**) {  
 peekParent = peek;  
 peek = peek.**leftChild**;  
 }  
 cursor.**data** = peek.**data**;  
 *//then delete the successor* peekParent.**leftChild** = peek.**rightChild**;  
 System.***out***.println(**"Value "** + a + **" deleted."**);  
 }  
  
 **private static class** Node {  
 **private int data**;  
 **private** Node **leftChild**;  
 **private** Node **rightChild**;  
 **private** Node **root**;  
  
 **public** Node() {  
 **leftChild** = **null**;  
 **rightChild** = **null**;  
 }  
  
 **public** Node(**int** a) {  
 **data** = a;  
 **leftChild** = **null**;  
 **rightChild** = **null**;  
 }  
 }  
}

**Sample Input/Output**

Welcome to the Binary Search Tree. What would you like to do?

Enter A for add, D for delete, P for print, Q for quit.

A

What number would you like to add?

7

Value 7 added.

Enter A for add, D for delete, P for print, Q for quit.

A

What number would you like to add?

3

Value 3 added.

Enter A for add, D for delete, P for print, Q for quit.

A

What number would you like to add?

2

Value 2 added.

Enter A for add, D for delete, P for print, Q for quit.

A

What number would you like to add?

3

Value 3 is already in tree.

Enter A for add, D for delete, P for print, Q for quit.

P

Preorder traversal:

7, 3, 2,

Inorder traversal:

2, 3, 7,

Postorder traversal:

2, 3, 7,

Enter A for add, D for delete, P for print, Q for quit.

D

What number would you like to delete?

2

Value 2 deleted.

Enter A for add, D for delete, P for print, Q for quit.

D

What number would you like to delete?

8

Value 8 is not in the tree.

Enter A for add, D for delete, P for print, Q for quit.

P

Preorder traversal:

7, 3,

Inorder traversal:

3, 7,

Postorder traversal:

3, 7,

Enter A for add, D for delete, P for print, Q for quit.

Q

Thank you for using the tree. Good bye.