B14_IN 2110 - Data Structures and Algorithms
ASSIGNMENT

Submission Date: Submit on or before January 23, 2017.

1. The following incomplete Java classes are written to implement a "STACK" using a link list.

```
class Node {
   int data;
   Node next;

   public Node(int i) {
   //A constructor to initialize members data and next
   }
   public void display() {
   //A method to display the data in the node
   }
}

class Stack {
   private Node top; //holds a reference to the top node

   public Stack() {
   //A constructor to initialize top
   }

   public boolean isEmpty() {
   // A method to check the stack is empty or not
   }

   public void push(int i) {
   // A method to push an 'int' i onto the stack
   }

   public Node pop() {
   //A method to remove the top node and returns the reference of the removed node
   }

   public Node peek() {
   //A method to peek the top node
   }
}
```

Write appropriate codes for the incomplete constructors/methods given in the above class as follows:

(i) A constructor to `Node` class that takes in an `int` parameter `i`. This constructor should initialize the members of `Node` class.

(ii) A method to display the contents of `Node` class.

(iii) A constructor to `STACK` class to initialize `top`.

(iv) A boolean method `isEmpty` to test whether the stack is empty or not.

(v) A `push` operator to insert an `int i` to the top of the stack.

(vi) A `pop` operator to remove the top node of the stack and returns the reference of the removed node. If the stack is empty the method should return a `null` reference.

(vii) A `peek` operator to obtain the value at top of the stack.

(viii) Write a suitable Java application class to convert a decimal number into the equivalent binary number. Use the above `Stack` class appropriately.

2. Consider the following incomplete Java classes that are written to obtain the mirror image of a binary search tree (BST).

```
class Node {

    int data;
    Node left;
    Node right;

    public Node(int i) {
        data = i;
        left = right = null;
    }

    public void display() {
    // Displays the node's content and its children's contents
    }

    public void swapChildren (){
    // Swaps the left and right childen of the node
    }
}

class Tree {

    private Node root;

    public Tree() {
        root = null;
    }

    public void insert(int i) {
    // Inserts int 'i' to the BST
    }

    public void printPreOrder (Node localRoot) {
```

```
    // Prints elements and their children's contents of a sub-tree with the root node '
     localRoot' using the pre-order traversal method
    }

    public void mirrorSubTree (Node localRoot){
    // Converts a sub-tree with the root node 'localRoot' to its mirror image
    }
}

public class MirrorImageApp {
    public static void main(String[] args) {
    // See Part (vi)
    }
}
```

Complete the codes of the following:

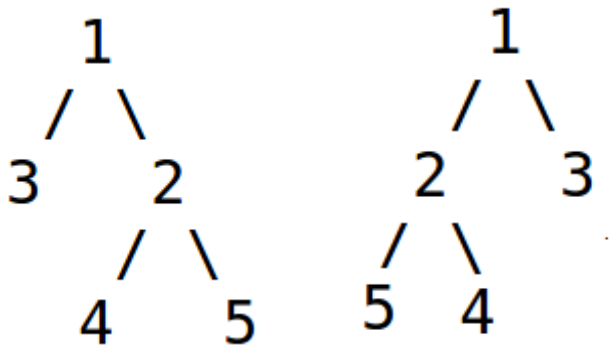
(i) `display ()` method to display the node's content and its children's contents in the following format:
`N = Node's content, L = Left child's content, R = Right child's content`
If there is no left or right child, display() method prints "NULL" in the respective place.

(ii) `swapChildren()` method to swap the left and right children of the node.

(iii) `insert()` method to insert an int 'i' to the BST.

(iv) A recursive method `printPreOrder()` to print elements and their children's contents of a sub-tree with the root node 'localRoot' using the pre-order traversal method. Use display() method written in part (i) appropriately.

(v) A recursive method `mirrorSubTree()` that converts a sub-tree with the root node 'localRoot' to its mirror image. An example is given in Fig. 1.



Original tree Mirror image of the tree

Figure 1: The mirror image of a tree

(vi) Write a suitable application class to test the methods written in parts (i) to (v).

Note: You are allowed to add additional methods to above classes appropriately. Use meaningful names to each of them and describe those methods as comments.

*********END*********