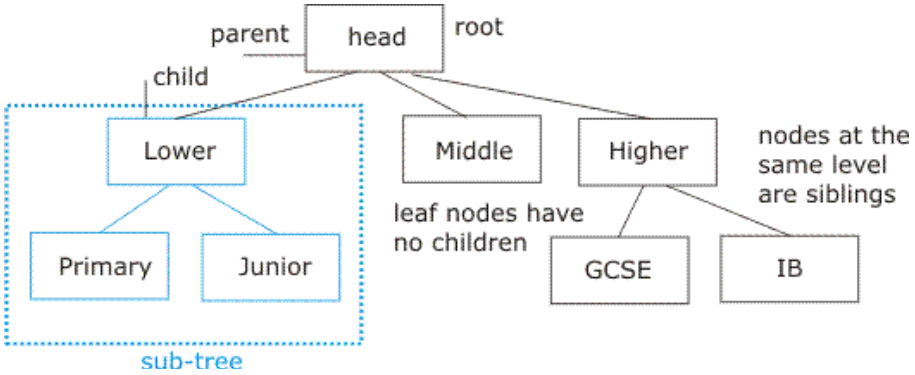
Binary Tree

The abstract data types (ADT's) we have seen so far (lists, stacks and queues) have been **linear** , that is each item is preceded or followed by exactly one item (unless, of course, it is the first or last data item).

Logical Structure

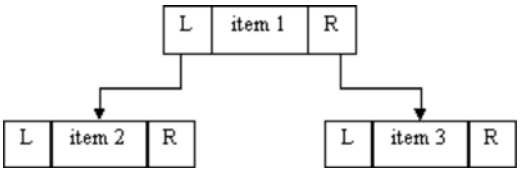
Trees are **non-linear hierarchical** data structures where every component may have **super-components** above and **sub-components** below.

A tree can also be described as a finite set of one or more linked **nodes** such that there is a special node with no parent called the **root** . Each **subtree** of the node is a **child** . Nodes at the same level are **siblings** . In either case a tree is probably a familiar concept already:



Looking at the sub tree we can see that it has all of the features of the main tree (a root, nodes and leaves) so that a tree is a **recursive** structure.

A binary tree is a collection of nodes, which has, at most, two children. It is relatively simple to implement such a structure using a node with a data field and two pointer fields:



Back to top

Binary Tree Node

As we did previously, we establish a class to represent the data structure and another to manipulate it:

```
 /**
* This class implements a binary tree node
* It's data field is of the type Object - which means it can store
```

In this course, we deal only with **Binary Trees** - a special kind of tree in which each node can have, at most, two children usually referred to as left-child and right-child).

Object is the super class of all classes created in Java.

You don't have any choice over that.

We have a left link and a right link.

If you understood the way linear Nodes worked in a simple linked list this should be a breeze.

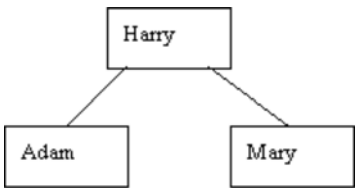
.

```
* data of any type
*/
public class Node
{
   private Node left;
   private Node right;
   private Object data;
   /**
   * Node constructor which sets the fields
   */
   public Node(Node lf, Node rt, Object dt)
   {
     setLeft( lf );
     setRight( rt );
     setData( dt );
   }
   // Accessor and mutator methods
   public Node getLeft() { return this.left; }
   public Node getRight() { return this.right; }
   public Object getData() { return this.data; }
   public void setLeft(Node n) { this.left = n; }
   public void setRight(Node n) { this.right = n; }
   public void setData(Object d) { this.data = d; }
}
```

Binary Tree Example

The class which demonstrates the binary tree is not particularly sophisticated.

This class enables the user to add items to the binary tree such that items can be retrieved in alphabetical order. For example the root node might be Harry. If the next node added is greater than Harry it is added to the right branch (say it is Mary), otherwise to the left (Adam):




```
 import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/**
* This class demonstrates use of a binary tree to store names
*/
public class BinaryTreeExample extends Applet implements ActionListener
{
  // interface objects
   Label dataLabel = new Label("Data:");
   TextField dataField = new TextField("Enter data here");

   Button addButton = new Button("Add item");
   Button newButton = new Button("New tree");
   Button preButton = new Button("Pre-order");
   Button inoButton = new Button("In-order");
   Button posButton = new Button("Post-order");

   TextArea display = new TextArea(10, 20);
  Label messages = new Label("System messages appear here");
```

Objects to enter data

Buttons for program options

display area

The usual sort of thing

if something was typed in, add it to the tree.

This is a simplistic approach since we have left

```
// pointer to root node
Node root;

/**
* Method to lay out the GUI and init the Applet
*/
public void init()
{
  add( dataLabel );
  add( dataField );
  // Button panels
  Panel actionButtons = new Panel();
  actionButtons.add( addButton );
  actionButtons.add( newButton);
  add(actionButtons);
  Panel displayButtons = new Panel();
  displayButtons.add( preButton );
  displayButtons.add( inoButton );
  displayButtons.add( posButton );
  add(displayButtons);
  add( display );
  add( messages );

  addButton.addActionListener(this);
  newButton.addActionListener(this);
  preButton.addActionListener(this);
  inoButton.addActionListener(this);
  posButton.addActionListener(this);

  root = null;
}
/**
* Detect and act on Button presses
*
* @param the event that caused the method to be called
*/
public void actionPerformed(ActionEvent e)
{
  display.setText("");
  if (e.getSource() == addButton)
  {
    // check for text in box
    String data = dataField.getText();
    if (data.length() != 0)
    {
      addNode(data);
    }
    else
    {
      messages.setText("No data to add!");
    }
  }
  else
  {
    // start a new tree
    if (e.getSource() == newButton)
    {
      root = null;
    }
    else
    {
      // a display button was pressed
      if( root != null )
      {
```

some nodes in memory by doing this.

We pass on the Button ID to another method.

```
          displayTree( e );
        }
        else
        {
          messages.setText("No tree to display!");
        }
      }
    }
}
/**
* Creates a new data Node and calls insertNode to actually put
* it in the tree
*
* @param the data to be added
*/
public void addNode( String data )
{
  // creates a new node to add
  Node temp = new Node(null, null, data);
 if ( root == null )
  {
    display.append("Starting new tree\n");
    root = temp;
  }
  else
  {
    // call recursive routine to insert node
    insertNode( root, temp );
  }
}
/**
* insert a node into the tree at the correct position
*
* @param n - the current point of the tree being examined
* @param temp - the Node to be added
*/
public void insertNode(Node n, Node temp)
{
  // get the data entries of the nodes,
  // the (String) is a cast - it converts type Object to type String
  String ns = (String) n.getData();
  String ts = (String) temp.getData();

  // compare the data elements
  if (ts.compareTo(ns) > 0)
  {
    // test add at right
    display.append("Testing right of " + ns + "\n");
    if (n.getRight() == null)
    {
      // found a space to put this node
      n.setRight(temp);
    }
    else
    {
      // try again at the next one down
      insertNode( n.getRight(), temp);
    }
  }
  else
  {
    // test add at left
    display.append("Testing left of " + ns + "\n");
```

Tree traversal is a topic we cover on the next page .

Visit left subtree

```
      if (n.getLeft() == null)
      {
        n.setLeft(temp);
      }
      else
      {
        insertNode( n.getLeft(), temp);
      }
    }
  }
 /**
  * We can display the tree in various ways
  *
  * @param e - the Button that triggered the display request
  */
  public void displayTree(ActionEvent e)
  {
    // selects display method depending upon button
    // pressed by user.
    if( e.getSource() == preButton )
    {
      displayTreePreOrder( root );
    }
    else
    {
      if( e.getSource() == inoButton )
      {
        displayTreeInOrder( root );
      }
      else
      {
        displayTreePostOrder( root );
      }
    }
  }
  /**
  * Inorder traversal of the tree
  *
  * @param n - the currently processed Node
  */
  public void displayTreeInOrder(Node n)
  {
    if (n != null)
    {
      displayTreeInOrder(n.getLeft());
      display.append((String) n.getData() + "\n");
      displayTreeInOrder(n.getRight());
    }
  }
  /**
  * Postorder traversal of the tree
  *
  * @param n - the currently processed Node
  */
  public void displayTreePostOrder(Node n)
  {
    if (n != null)
    {
      displayTreePostOrder(n.getLeft());
      displayTreePostOrder(n.getRight());
      display.append((String) n.getData() + "\n");
    }
  }
```

output data
visit right sub-tree

Visit left subtree
visit right sub-tree
output data

output data
Visit left subtree
visit right sub-tree

Hint : s1.compareTo (s2) evaluates to a 0 if both Strings are equal.

```
   /**
   * Preorder traversal of the tree
   *
   * @param n - the currently processed Node
   */
   public void displayTreePreOrder(Node n)
   {
     if (n != null)
     {
       display.append((String) n.getData() + "\n");
       displayTreePreOrder(n.getLeft());
       displayTreePreOrder(n.getRight());
     }
   }
}
```

Back to top

Exercise

What happens if you try to enter the same data item more than once? Trace the insertNode algorithm to find out where that happens, prevent it and output a suitable error message .

Modify the example to store a simple data object instead of a String.

Back to top

Related: [Java home | **Previous**: recursion | **Next:** tree traversal]