Trees

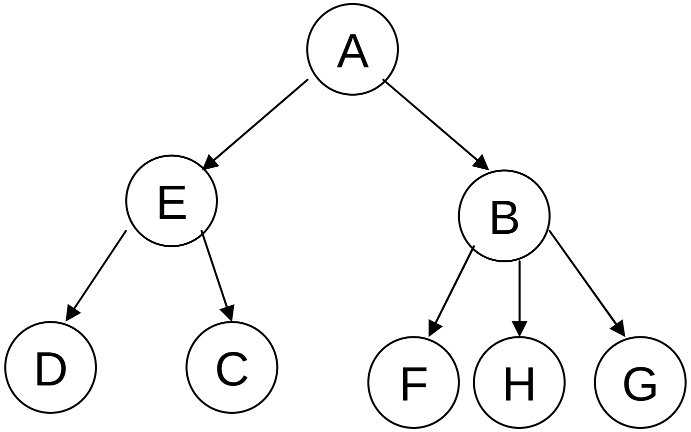
- What is a Tree?
- Tree terminology
- Why trees?
- What is a general tree?
- Implementing trees
- Binary trees
- Binary tree implementation
- Application of Binary trees

What is a Tree?

- A tree, is a finite set of nodes together with a finite set of directed edges that define parent-child relationships. Each directed edge connects a parent to its child. Example:

Nodes={A,B,C,D,E,f,G,H}

Edges={(A,B),(A,E),(B,F),(B,G),(B,H), (E,C),(E,D)}

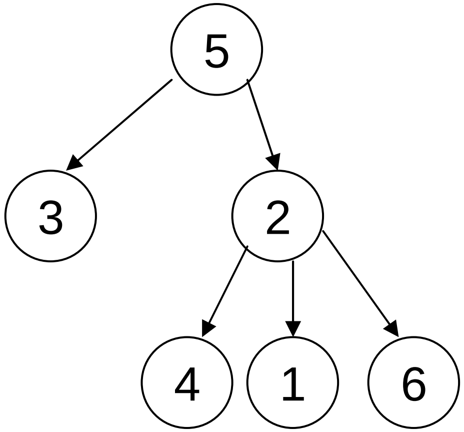


- A directed **path** from node m1 to node m_k is a list of nodes m_1, m_2, . . . , m_k such that each is the parent of the next node in the list. The length of such a path is k - 1.
- Example: A, E, C is a directed path of length 2.

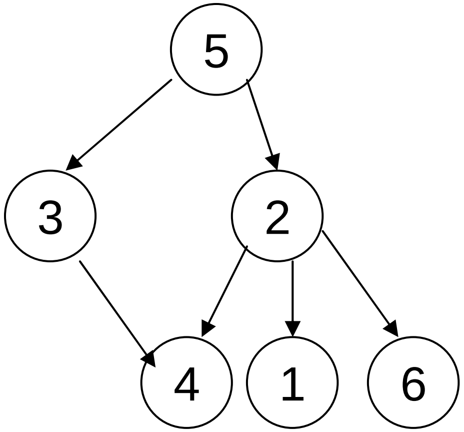
What is a Tree? (contd.)

- A tree satisfies the following properties:

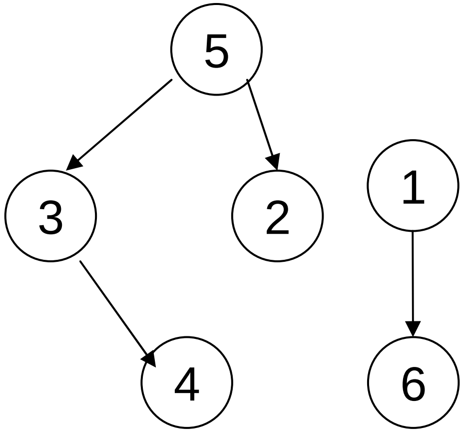
1. It has one designated node, called the root, that has no parent.
2. Every node, except the root, has exactly one parent.
3. A node may have zero or more children.
4. There is a unique directed path from the root to each node.



tree



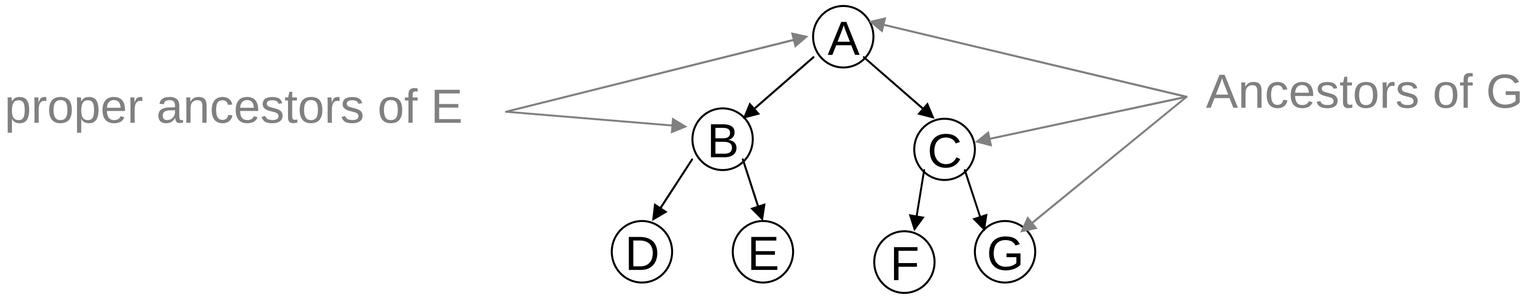
Not a tree



Not a tree

Tree Terminology

- Ordered tree: A tree in which the children of each node are linearly ordered (usually from left to right).

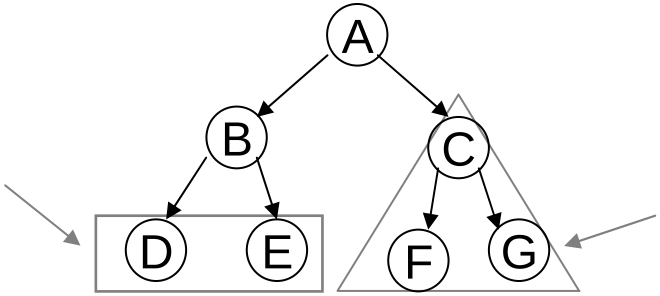


An Ordered Tree

- **Ancestor** of a node v: Any node, including v itself, on the path from the root to the node.
- **Proper ancestor** of a node v: Any node, excluding v, on the path from the root to the node.

Tree Terminology (Contd.)

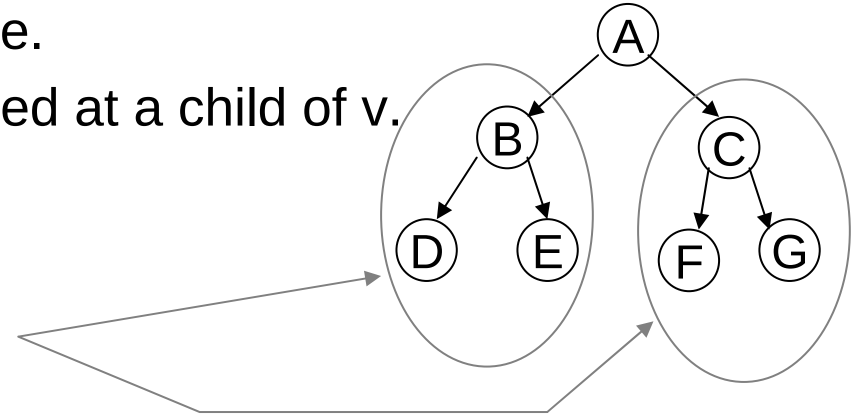
- **Descendant** of a node v: Any node, including v itself, on any path from the node to a leaf node (i.e., a node with no children).



Proper descendants of node B

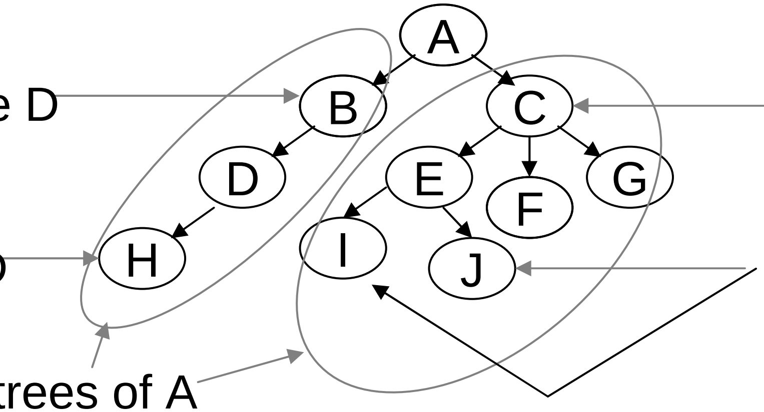
Descendants of a node C

- **Proper descendant** of a node v: Any node, excluding v, on any path from the node to a leaf node.
- **Subtree** of a node v: A tree rooted at a child of v.



subtrees of node A

Tree Terminology (Contd.)



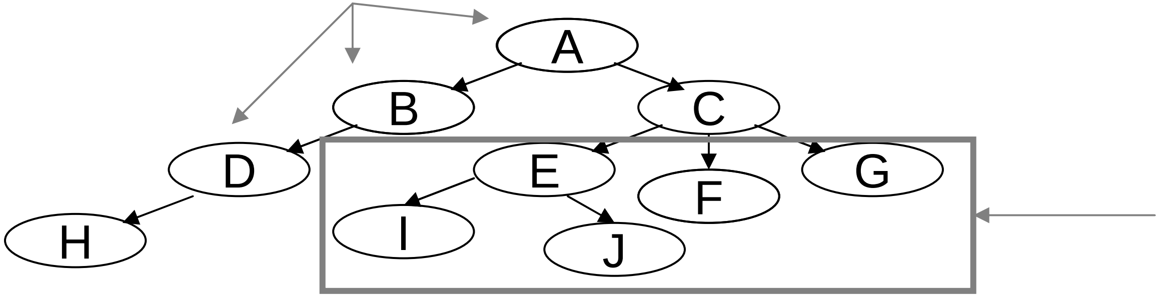
parent of node D

grandfather of nodes I,J

child of node D

grandchildren of node C

subtrees of A



proper ancestors of node H

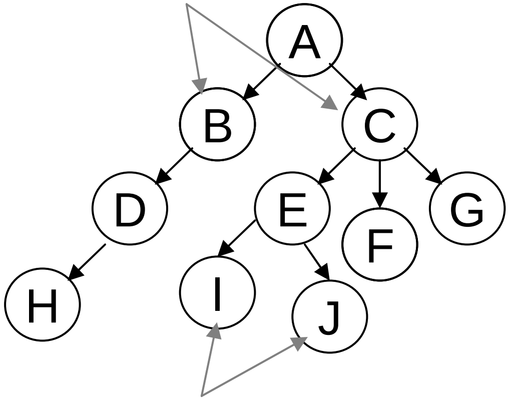
proper descendants of node C

Tree Terminology (Contd.)

- Degree: The number of subtrees of a node
  - Each of node D and B has degree 1.
  - Each of node A and E has degree 2.
  - Node C has degree 3.
  - Each of node F,G,H,I,J has degree 0.

An Ordered Tree
with size of 10

Siblings of A



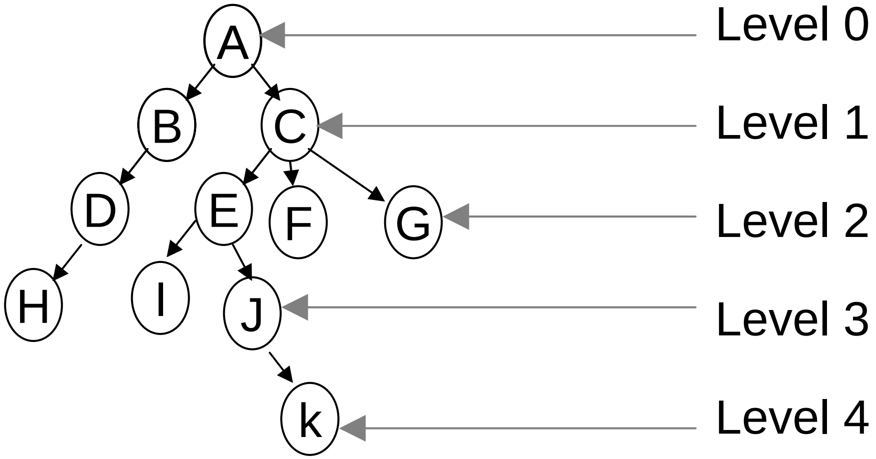
Siblings
of E

- **Leaf**: A node with degree 0.
- **Internal** or interior node: a node with degree greater than 0.
- **Siblings**: Nodes that have the same parent.
- **Size**: The number of nodes in a tree.

Tree Terminology (Contd.)

- **Level** (or depth) of a node v: The length of the path from the root to v.
- **Height** of a node v: The length of the longest path from v to a leaf node.
  - The height of a tree is the height of its root mode.
  - By definition the height of an empty tree is -1.

- The height of the tree is 4.
- The height of node C is 3.



Why Trees?

- Trees are very important data structures in computing.
- They are suitable for:
  - Hierarchical structure representation, e.g.,
    - File directory.
    - Organizational structure of an institution.
    - Class inheritance tree.
  - Problem representation, e.g.,
    - Expression tree.
    - Decision tree.
  - Efficient algorithmic solutions, e.g.,
    - Search trees.
    - Efficient priority queues via heaps.

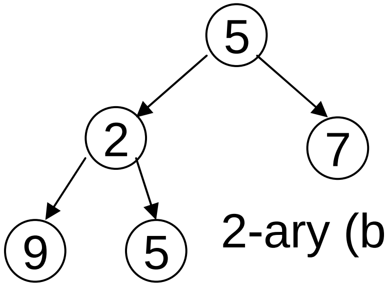
General Trees and its Implementation

- In a general tree, there is no limit to the number of children that a node can have.
- Representing a general tree by linked lists:
  - Each node has a linked list of the subtrees of that node.
  - Each element of the linked list is a subtree of the current node

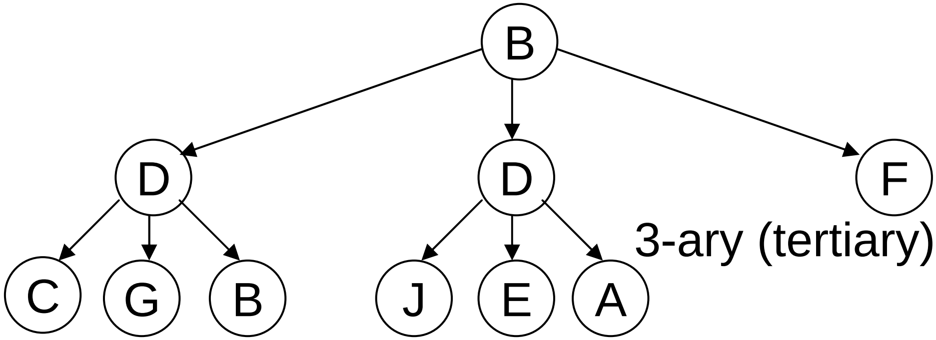
```
public class GeneralTree extends AbstractContainer {
   protected Object key ;
   protected int degree ;
   protected MyLinkedList  list ;
   // . . .
}
```

N-ary Trees

- An N-ary tree is an ordered tree that is either:
  1. Empty, or
  2. It consists of a root node and at most N non-empty N-ary subtrees.
- It follows that the degree of each node in an N-ary tree is at most N.
- Example of N-ary trees:



2-ary (binary) tree



3-ary (tertiary)tree

N-ary Trees Implementation

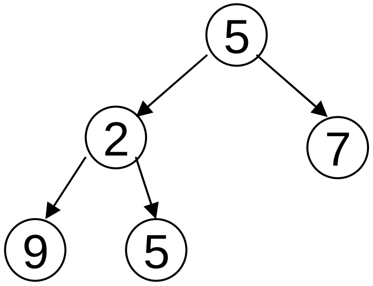
```
public class  NaryTree extends AbstractTree {
   protected  Object  key ;
   protected  int  degree ;
   protected  NaryTree[ ]  subtree ;

   public  NaryTree(int degree){
      key = null ; this.degree = degree ;
      subtree = null ;
   }

   public  NaryTree(int degree, Object key){
      this.key = key ;
      this.degree = degree ;
      subtree = new NaryTree[degree] ;
      for(int i = 0; i < degree; i++)
         subtree[i] = new NaryTree(degree);
   }
   // . . .
}
```

Binary Trees

- A binary tree is an N-ary tree for which N = 2.
- Thus, a binary tree is either:

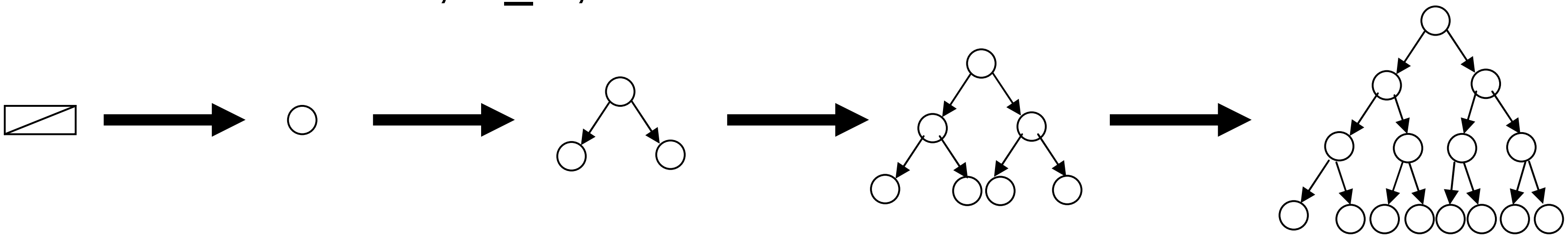
1. An empty tree, or
2. A tree consisting of a root node and at most two non-empty binary subtrees.

Example:



Binary Trees (Contd.)

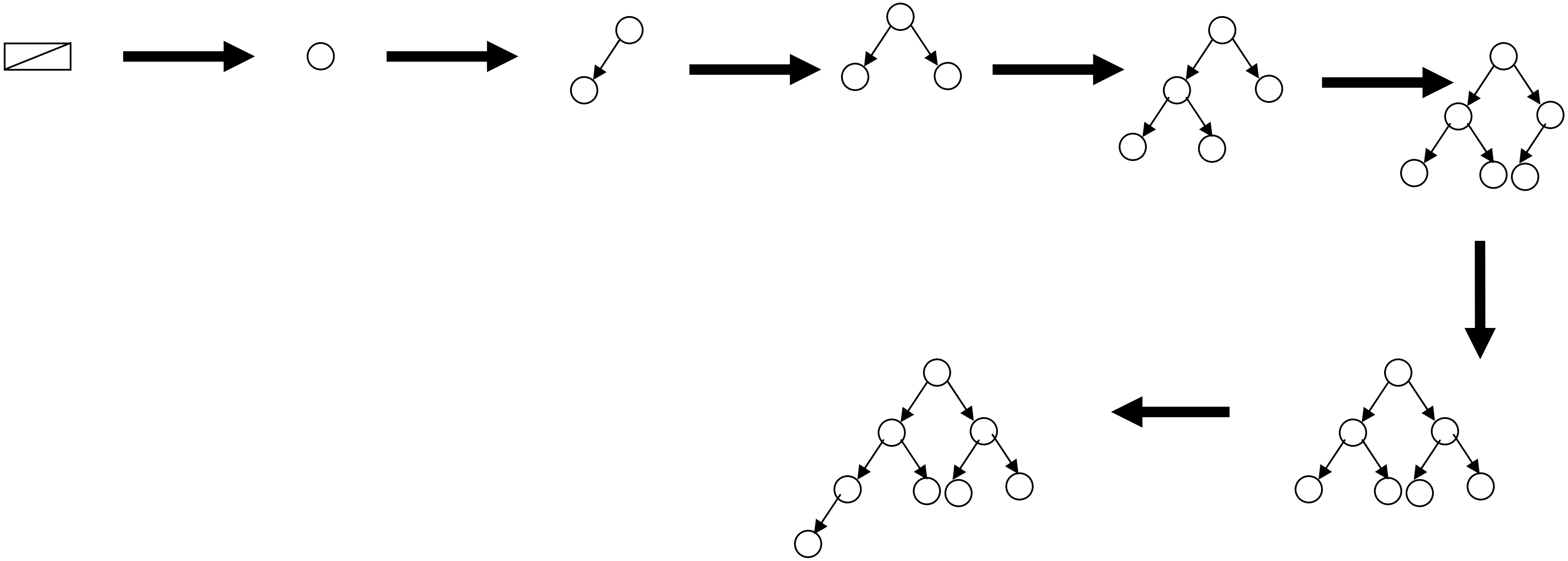
- A full binary tree is either an empty binary tree or a binary tree in which each level k, k $\geq$ 0, has 2^k nodes.



- A complete binary tree is either an empty binary tree or a binary tree in which:
  1. Each level k, k $\geq$ 0, other than the last level contains the maximum number of nodes for that level, that is 2^k.
  2. The last level may or may not contain the maximum number of nodes.
  3. If a slot with a missing node is encountered when scanning the last level in a left to right direction, then all remaining slots in the level must be empty.
- Thus, every full binary tree is a complete binary tree, but the opposite is not true.

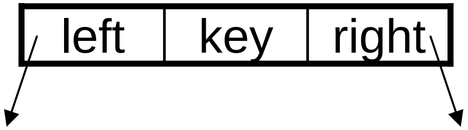
Binary Trees (Contd.)

- Example showing the growth of a complete binary tree:

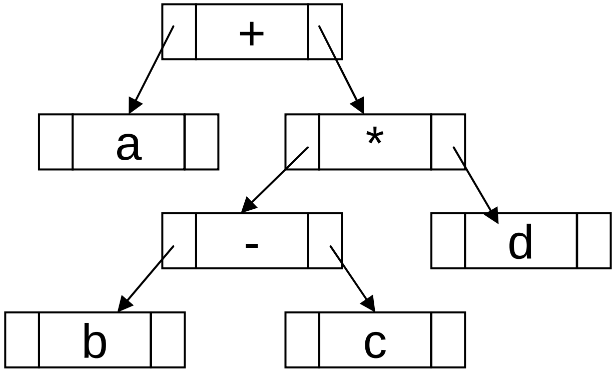


Binary Trees Implementation

```
public class BinaryTree
            extends AbstractTree{
   protected Object key ;
   protected BinaryTree  left, right ;
   public BinaryTree(Object key,
                        BinaryTree left,
                        BinaryTree right){
      this.key = key ;
      this.left = left ;
      this.right = right ;
   }
   public BinaryTree( ) {
      this(null, null, null) ;
   }
   public BinaryTree(Object key){
      this(key, new BinaryTree( ),
               new BinaryTree( ));
   }
   // . . .
}
```



Example: A binary tree representing a + (b - c) * d



Binary Trees Implementation (Contd.)

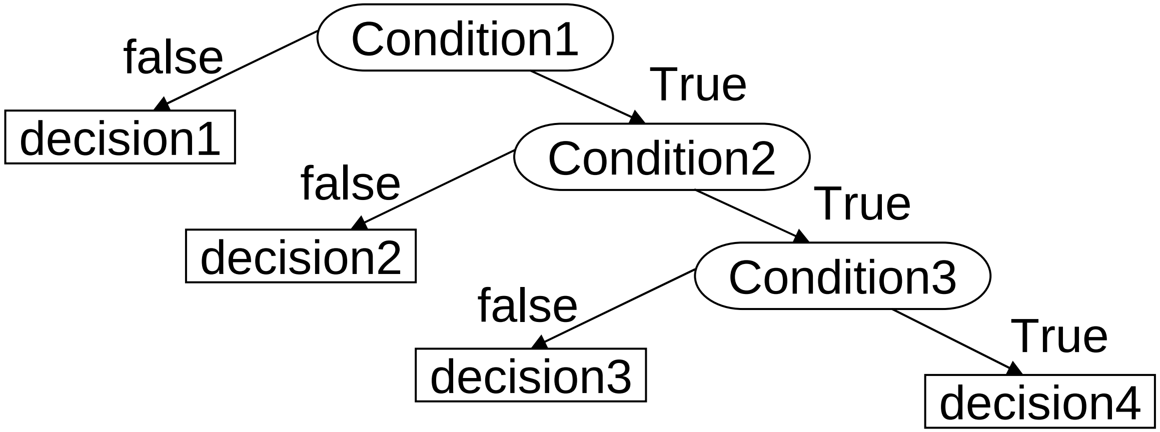
```
public boolean isEmpty( ){  return key == null ; }

public boolean isLeaf( ){
   return ! isEmpty( ) && left.isEmpty( ) && right.isEmpty( ) ; }

public Object getKey( ){
   if(isEmpty( )) throw new InvalidOperationException( ) ;
   else return key ;
}
public int  getHeight( ){
   if(isEmpty( )) return -1 ;
   else return 1 + Math.max(left.getHeight( ), right.getHeight( )) ;
}

public void attachKey(Object obj){
   if(! isEmpty( )) throw new InvalidOperationException( ) ;
   else{
      key = obj ;
      left = new BinaryTree( ) ;
      right = new BinaryTree( ) ;
   }
}
```

Binary Trees Implementation (Contd.)

```
public Object  detachKey( ){
   if(! isLeaf( )) throw new InvalidOperationException( ) ;
   else {
      Object obj = key ;
      key = null ;
      left = null ;
      right = null ;
      return obj ;
   }
}

public BinaryTree getLeft( ){
   if(isEmpty( )) throw new InvalidOperationException( ) ;
   else return left ;
}

public BinaryTree getRight( ){
   if(isEmpty( )) throw new InvalidOperationException( ) ;
   else return right ;
}
```

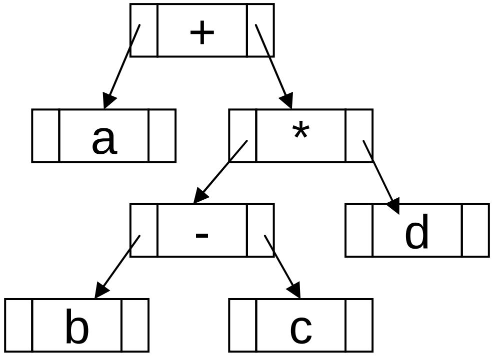
Application of Binary Trees

- Binary trees have many important uses. Two examples are:
  1. Binary decision trees.
     - Internal nodes are conditions. Leaf nodes denote decisions.



Condition1
false
decision1
True
Condition2
false
decision2
True
Condition3
false
decision3
True
decision4

- Expression Trees



+
a
*
-
d
b
c