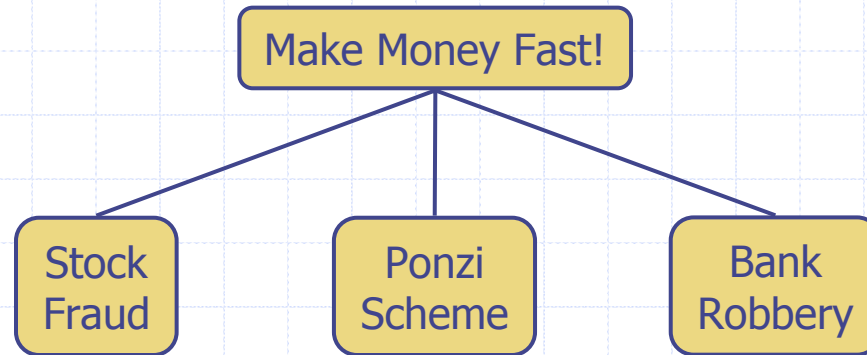


Trees

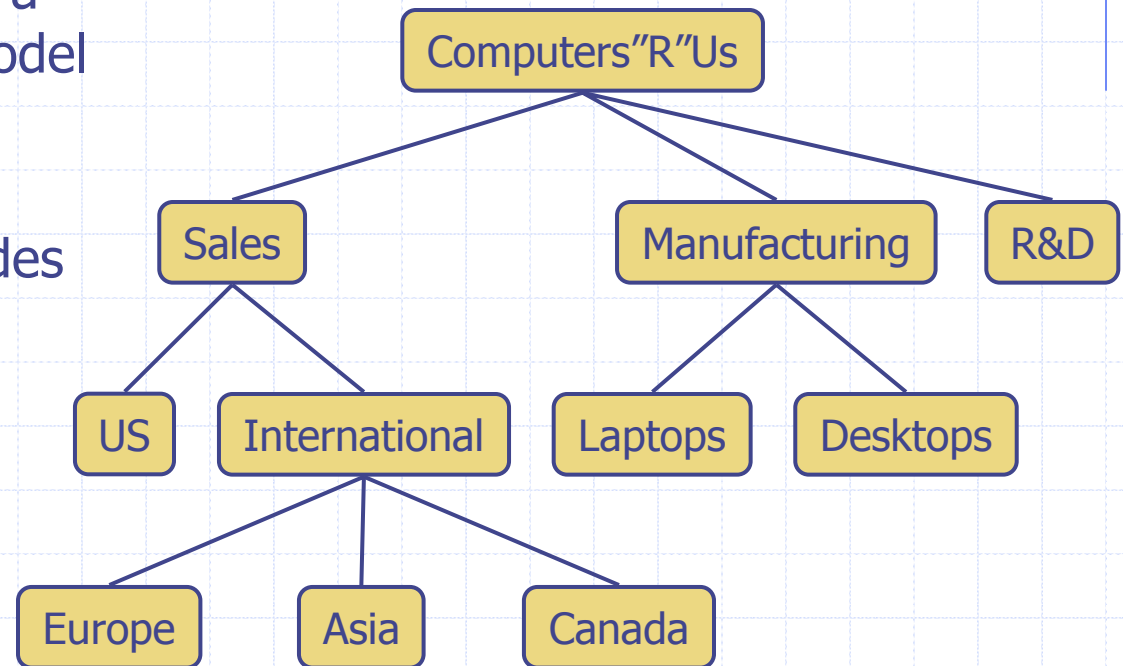


Outline and Reading

- ◆ Tree ADT (§2.3.1)
- ◆ Preorder and postorder traversals (§2.3.2)
- ◆ BinaryTree ADT (§2.3.3)
- ◆ Inorder traversal (§2.3.3)
- ◆ Euler Tour traversal (§2.3.3)
- ◆ Template method pattern
- ◆ Data structures for trees (§2.3.4)
- ◆ Java implementation (<http://jdsl.org>)

What is a Tree

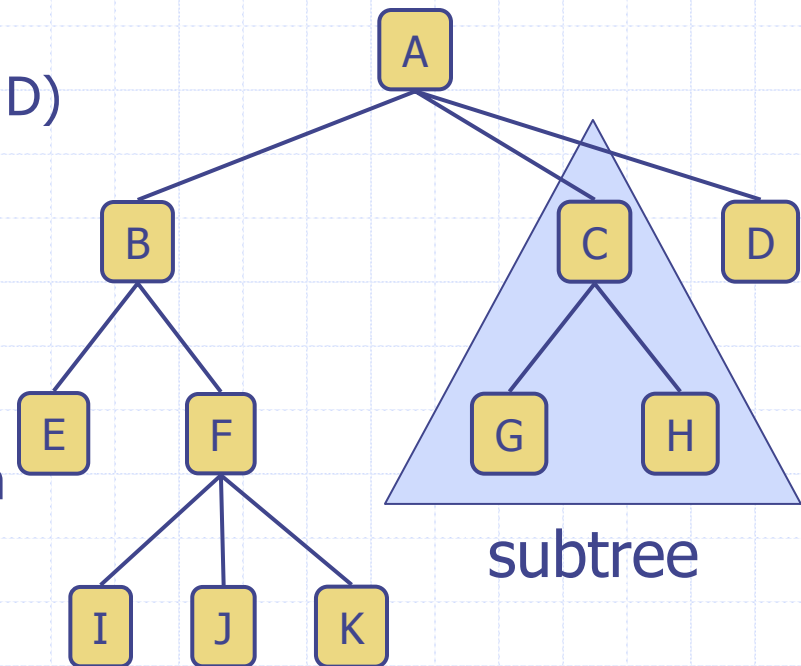
- ◆ In computer science, a tree is an abstract model of a hierarchical structure
- ◆ A tree consists of nodes with a parent-child relation
- ◆ Applications:
 - Organization charts
 - File systems
 - Programming environments



Tree Terminology

- ◆ Root: node without parent (A)
- ◆ Internal node: node with at least one child (A, B, C, F)
- ◆ External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- ◆ Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- ◆ Depth of a node: number of ancestors
- ◆ Height of a tree: maximum depth of any node (3)
- ◆ Descendant of a node: child, grandchild, grand-grandchild, etc.

- ◆ Subtree: tree consisting of a node and its descendants



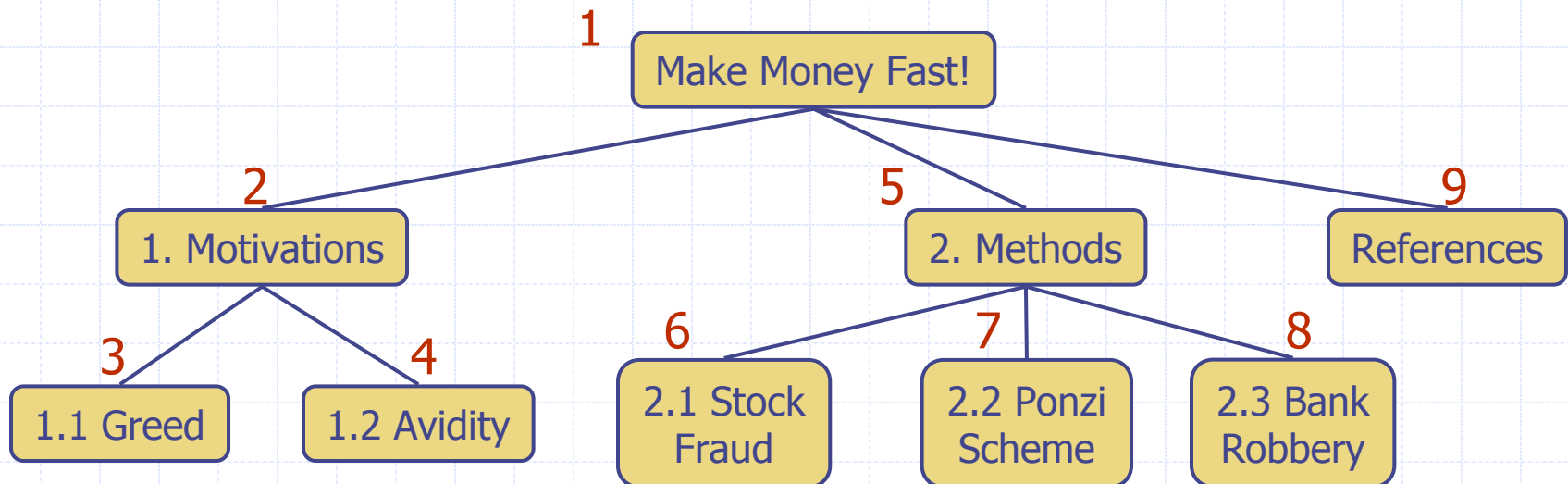
Tree ADT

- ◆ We use positions to abstract nodes
- ◆ Generic methods:
 - integer `size()`
 - boolean `isEmpty()`
 - objectIterator `elements()`
 - positionIterator `positions()`
- ◆ Accessor methods:
 - position `root()`
 - position `parent(p)`
 - positionIterator `children(p)`
- ◆ Query methods:
 - boolean `isInternal(p)`
 - boolean `isExternal(p)`
 - boolean `isRoot(p)`
- ◆ Update methods:
 - `swapElements(p, q)`
 - object `replaceElement(p, o)`
- ◆ Additional update methods may be defined by data structures implementing the Tree ADT

Preorder Traversal

- ◆ A traversal visits the nodes of a tree in a systematic manner
- ◆ In a preorder traversal, a node is visited before its descendants
- ◆ Application: print a structured document

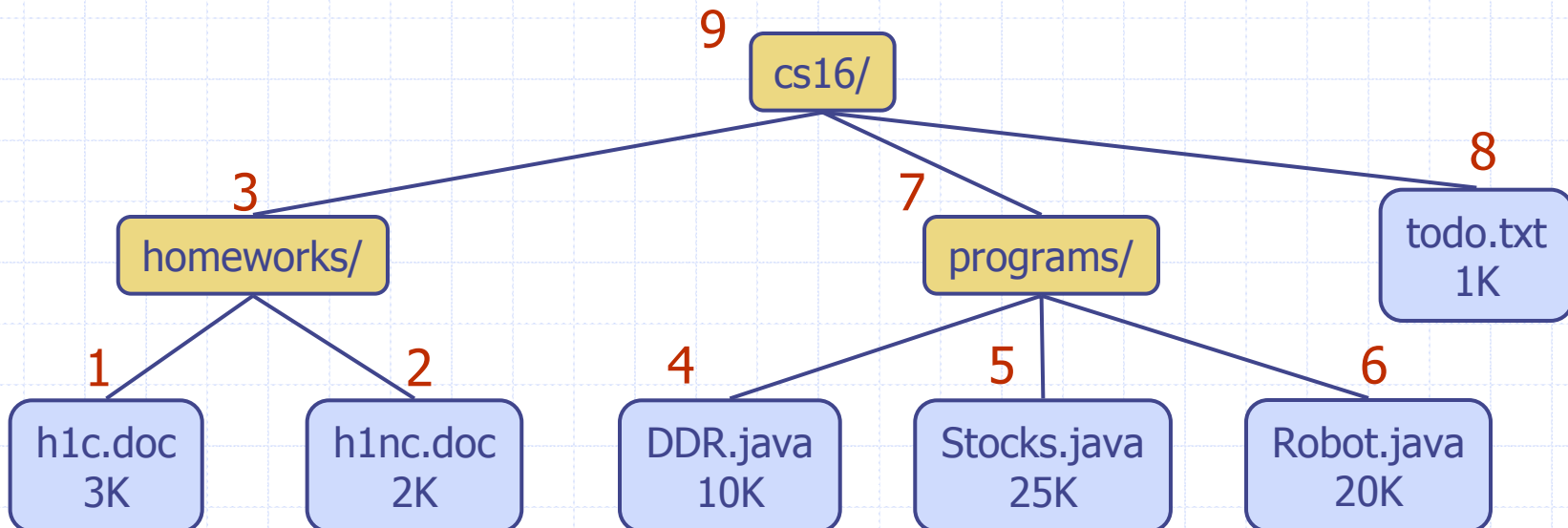
```
Algorithm preOrder(v)  
  visit(v)  
  for each child w of v  
    preorder(w)
```



Postorder Traversal

- ◆ In a postorder traversal, a node is visited after its descendants
- ◆ Application: compute space used by files in a directory and its subdirectories

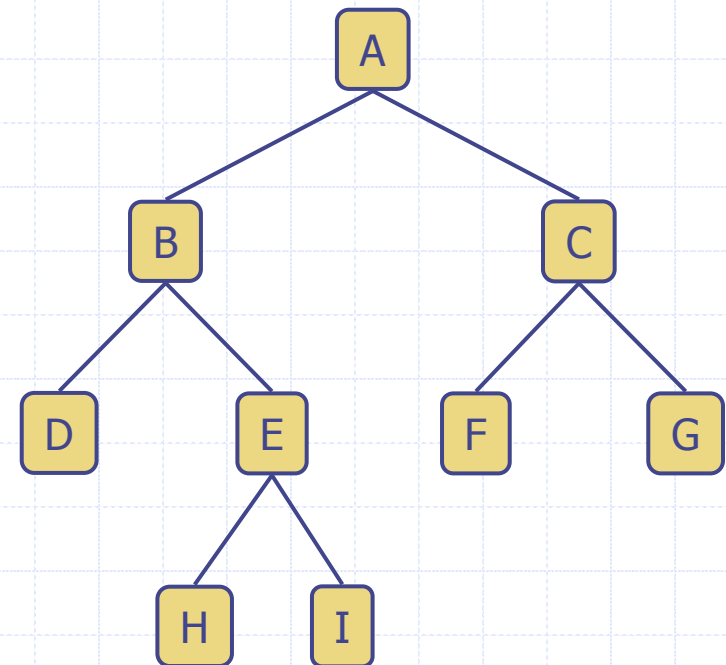
Algorithm *postOrder(v)*
for each child *w* of *v*
postOrder(w)
visit(v)



Binary Tree

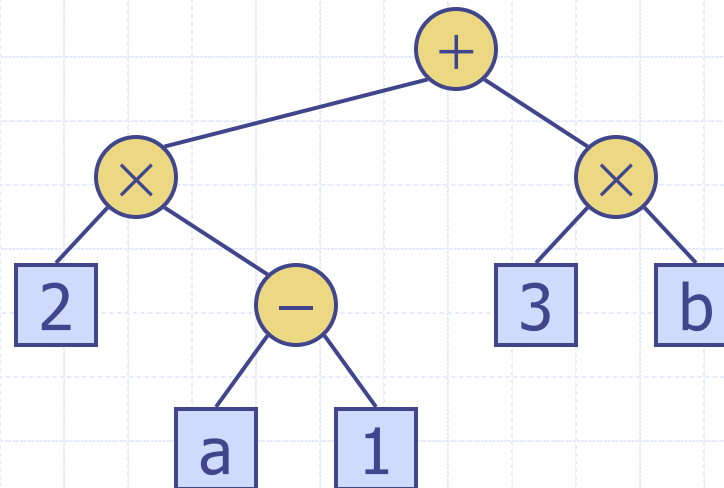
- ◆ A binary tree is a tree with the following properties:
 - Each internal node has two children
 - The children of a node are an ordered pair
- ◆ We call the children of an internal node left child and right child
- ◆ Alternative recursive definition: a binary tree is either
 - a tree consisting of a single node, or
 - a tree whose root has an ordered pair of children, each of which is a binary tree

- ◆ Applications:
 - arithmetic expressions
 - decision processes
 - searching



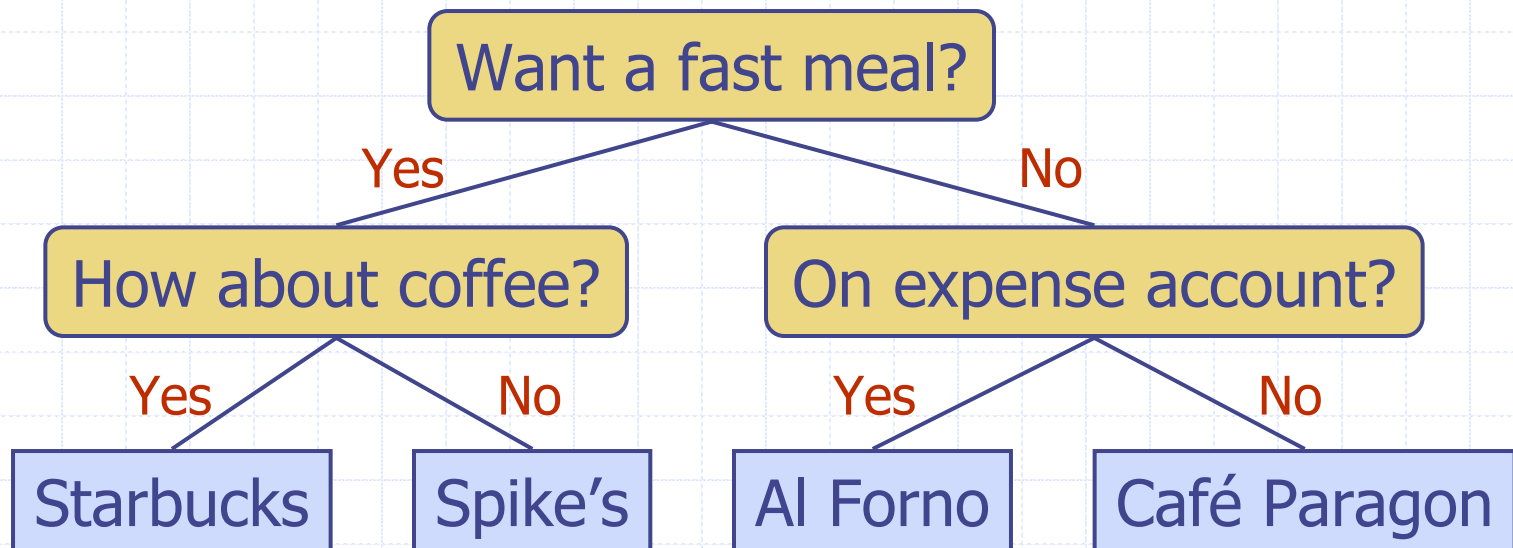
Arithmetic Expression Tree

- ◆ Binary tree associated with an arithmetic expression
 - internal nodes: operators
 - external nodes: operands
- ◆ Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



Decision Tree

- ◆ Binary tree associated with a decision process
 - internal nodes: questions with yes/no answer
 - external nodes: decisions
- ◆ Example: dining decision



Properties of Binary Trees

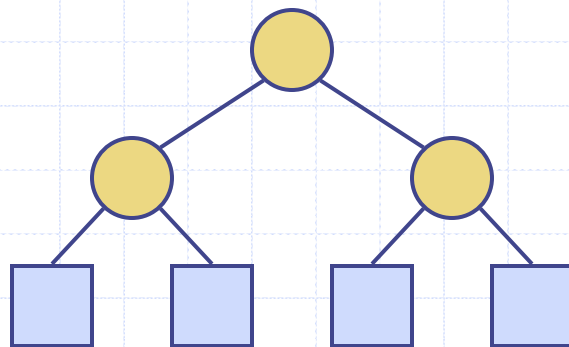
◆ Notation

n number of nodes

e number of external nodes

i number of internal nodes

h height



◆ Properties:

- $e = i + 1$

- $n = 2e - 1$

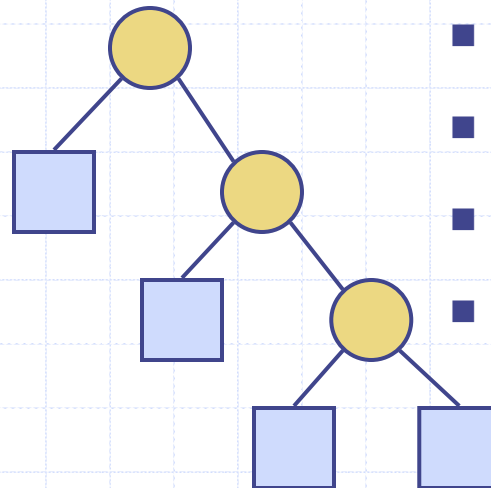
- $h \leq i$

- $h \leq (n - 1)/2$

- $e \leq 2^h$

- $h \geq \log_2 e$

- $h \geq \log_2 (n + 1) - 1$



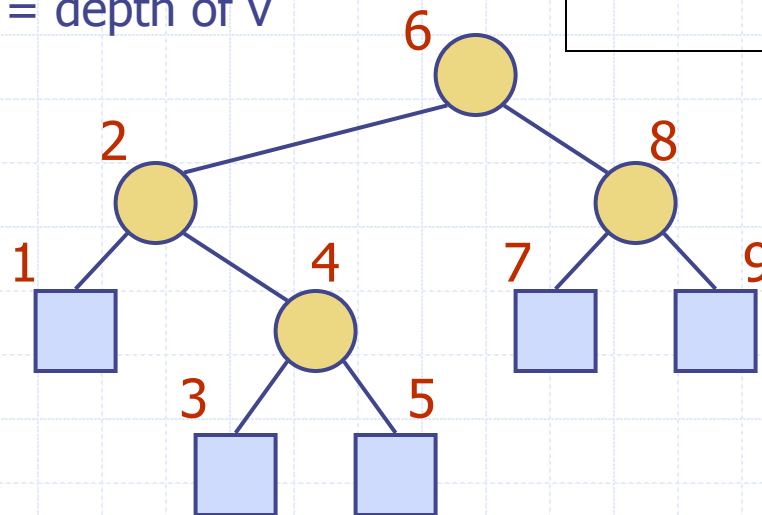
BinaryTree ADT

- ◆ The BinaryTree ADT extends the Tree ADT, i.e., it inherits all the methods of the Tree ADT
- ◆ Update methods may be defined by data structures implementing the BinaryTree ADT
- ◆ Additional methods:
 - position **leftChild**(p)
 - position **rightChild**(p)
 - position **sibling**(p)

Inorder Traversal

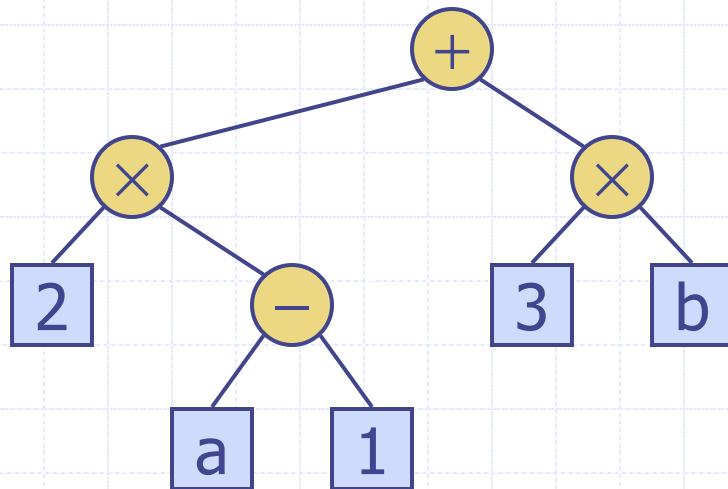
- ◆ In an inorder traversal a node is visited after its left subtree and before its right subtree
- ◆ Application: draw a binary tree
 - $x(v)$ = inorder rank of v
 - $y(v)$ = depth of v

```
Algorithm inOrder( $v$ )  
  if isInternal ( $v$ )  
    inOrder (leftChild ( $v$ ))  
  visit( $v$ )  
  if isInternal ( $v$ )  
    inOrder (rightChild ( $v$ ))
```



Print Arithmetic Expressions

- ◆ Specialization of an inorder traversal
 - print operand or operator when visiting node
 - print "(" before traversing left subtree
 - print ")" after traversing right subtree



Algorithm *printExpression(v)*

if *isInternal* (v)

print("(")

inOrder (*leftChild* (v))

print(v.*element* ())

if *isInternal* (v)

inOrder (*rightChild* (v))

print (")")

$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

- ◆ Specialization of a postorder traversal
 - recursive method returning the value of a subtree
 - when visiting an internal node, combine the values of the subtrees

Algorithm *evalExpr(v)*

if *isExternal*(v)

return *v.element*()

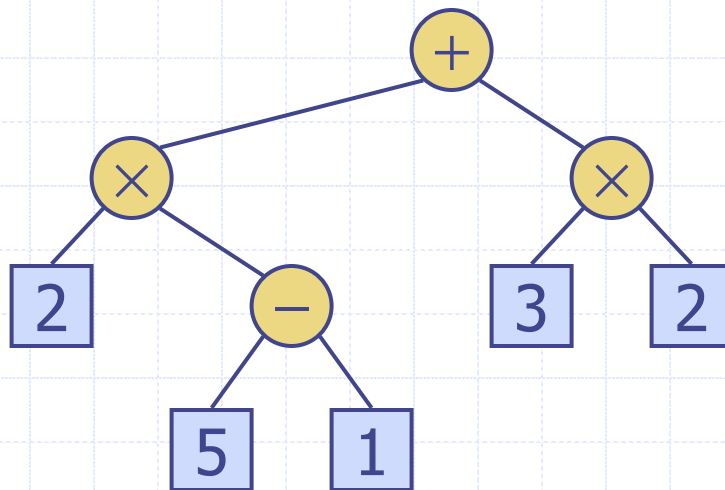
else

x ← *evalExpr*(*leftChild*(v))

y ← *evalExpr*(*rightChild*(v))

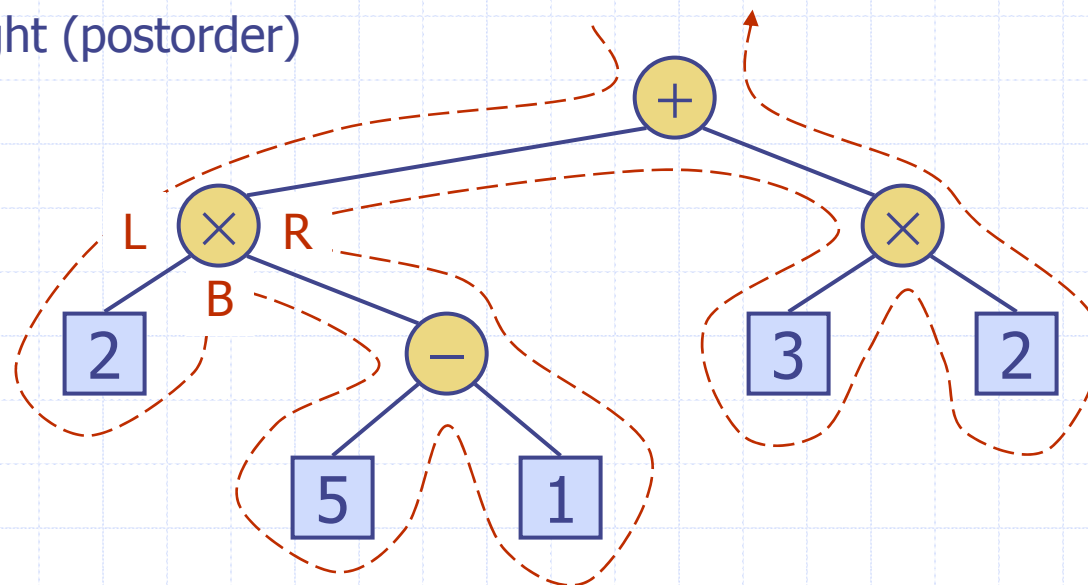
◇ ← operator stored at v

return *x* ◇ *y*



Euler Tour Traversal

- ◆ Generic traversal of a binary tree
- ◆ Includes a special cases the preorder, postorder and inorder traversals
- ◆ Walk around the tree and visit each node three times:
 - on the left (preorder)
 - from below (inorder)
 - on the right (postorder)



Template Method Pattern

- ◆ Generic algorithm that can be specialized by redefining certain steps
- ◆ Implemented by means of an abstract Java class
- ◆ Visit methods that can be redefined by subclasses
- ◆ Template method `eulerTour`
 - Recursively called on the left and right children
 - A `Result` object with fields `leftResult`, `rightResult` and `finalResult` keeps track of the output of the recursive calls to `eulerTour`

```
public abstract class EulerTour {
    protected BinaryTree tree;
    protected void visitExternal(Position p, Result r) {}
    protected void visitLeft(Position p, Result r) {}
    protected void visitBelow(Position p, Result r) {}
    protected void visitRight(Position p, Result r) {}
    protected Object eulerTour(Position p) {
        Result r = new Result();
        if tree.isExternal(p) { visitExternal(p, r); }
        else {
            visitLeft(p, r);
            r.leftResult = eulerTour(tree.leftChild(p));
            visitBelow(p, r);
            r.rightResult = eulerTour(tree.rightChild(p));
            visitRight(p, r);
            return r.finalResult;
        } ...
    }
}
```

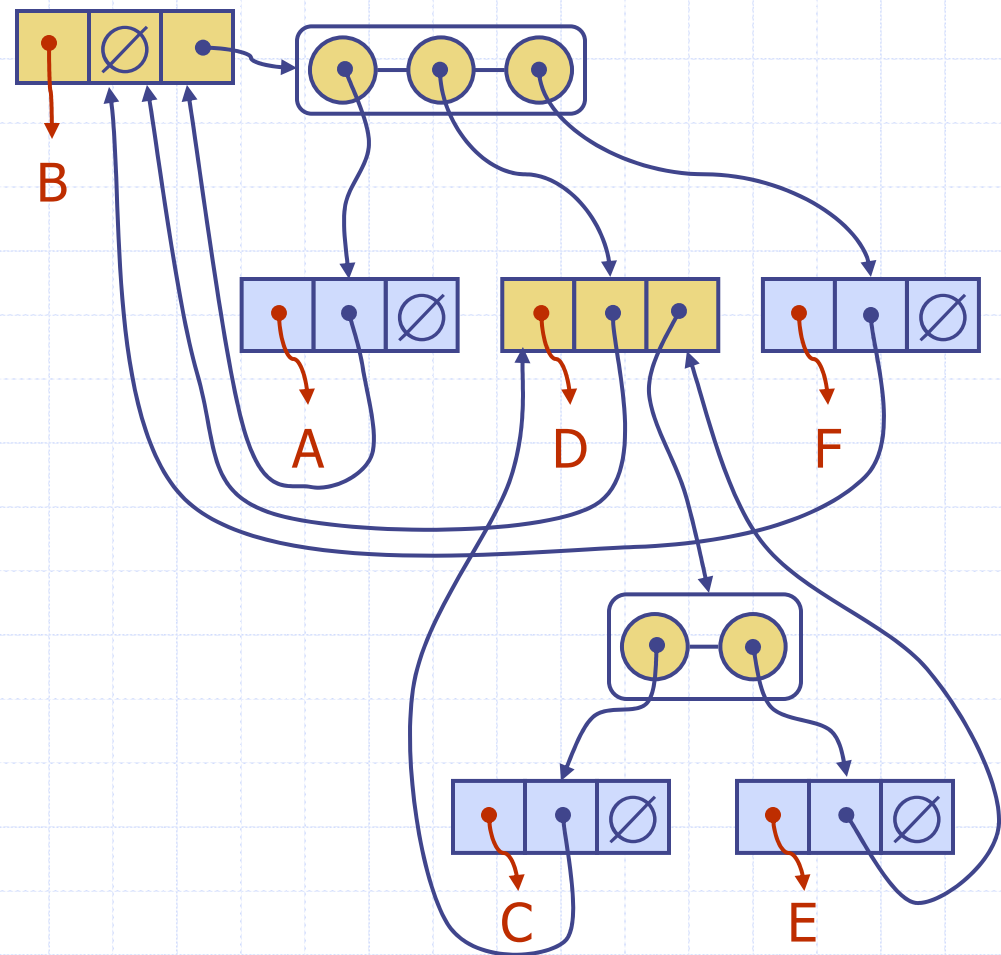
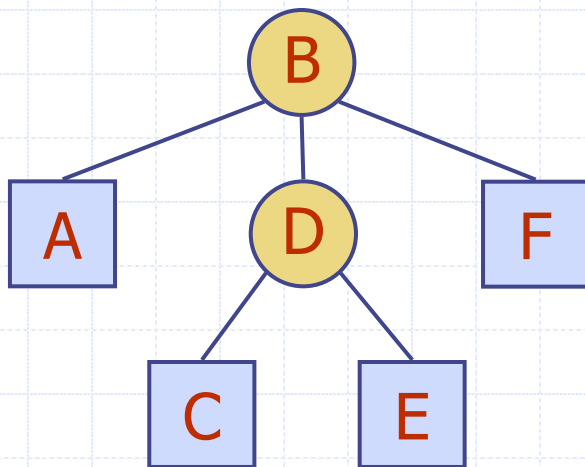
Specializations of EulerTour

- ◆ We show how to specialize class EulerTour to evaluate an arithmetic expression
- ◆ Assumptions
 - External nodes store Integer objects
 - Internal nodes store Operator objects supporting method operation (Integer, Integer)

```
public class EvaluateExpression
    extends EulerTour {
    protected void visitExternal(Position p, Result r) {
        r.finalResult = (Integer) p.element();
    }
    protected void visitRight(Position p, Result r) {
        Operator op = (Operator) p.element();
        r.finalResult = op.operation(
            (Integer) r.leftResult,
            (Integer) r.rightResult
        );
    }
    ...
}
```

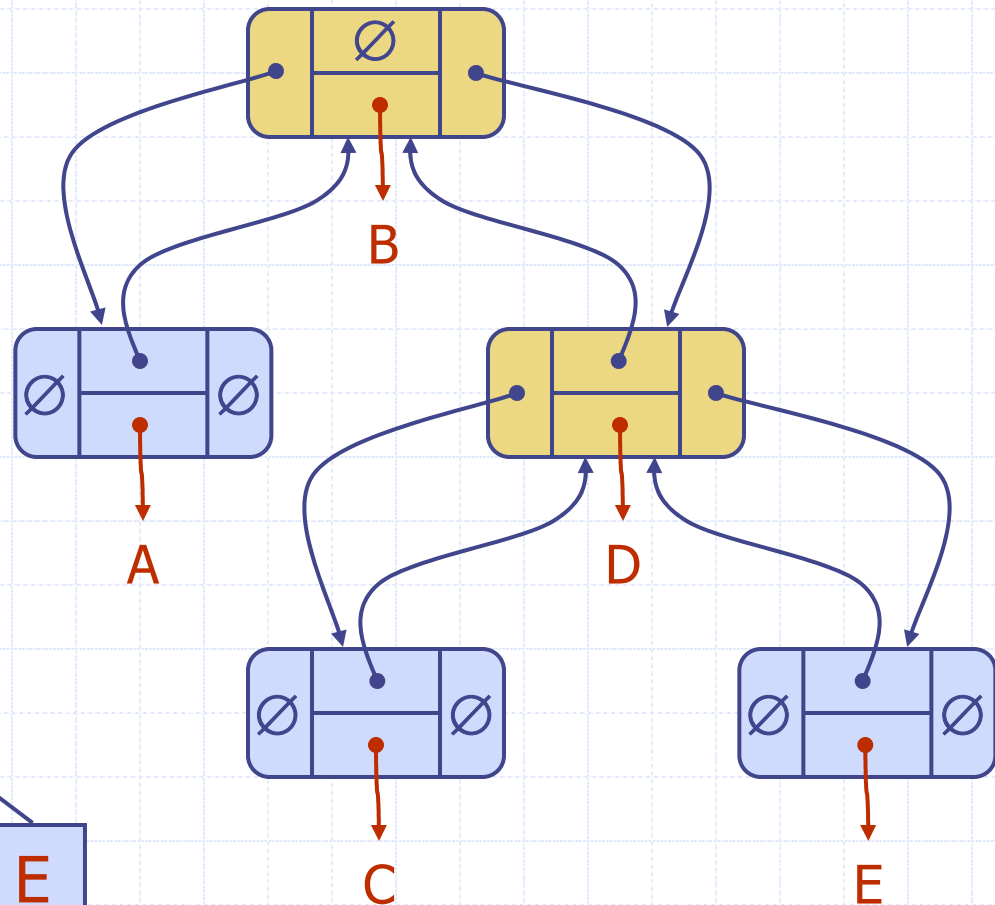
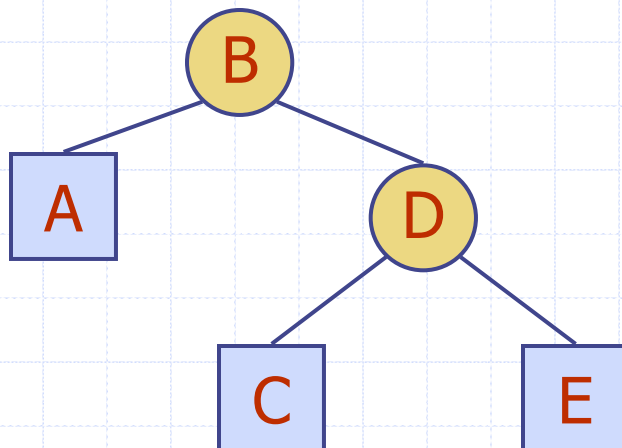
Data Structure for Trees

- ◆ A node is represented by an object storing
 - Element
 - Parent node
 - Sequence of children nodes
- ◆ Node objects implement the Position ADT



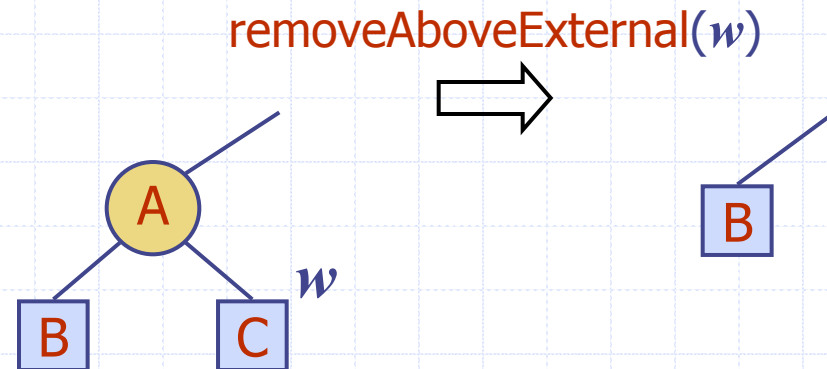
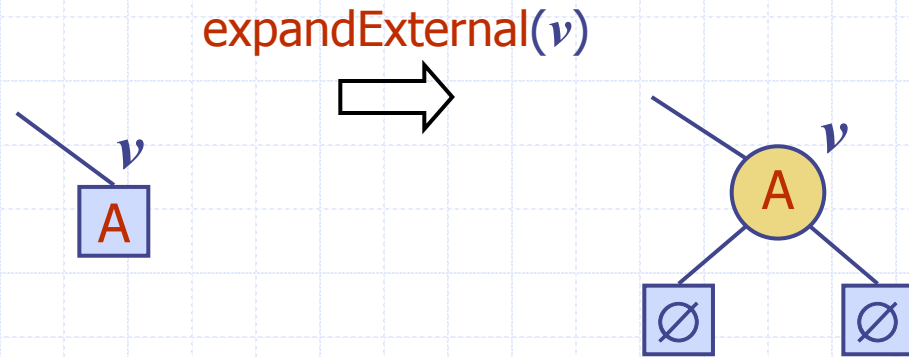
Data Structure for Binary Trees

- ◆ A node is represented by an object storing
 - Element
 - Parent node
 - Left child node
 - Right child node
- ◆ Node objects implement the Position ADT



Java Implementation

- ◆ Tree interface
- ◆ BinaryTree interface extending Tree
- ◆ Classes implementing Tree and BinaryTree and providing
 - Constructors
 - Update methods
 - Print methods
- ◆ Examples of updates for binary trees
 - `expandExternal(v)`
 - `removeAboveExternal(w)`



Trees in JDSL

- ◆ JDSL is the Library of Data Structures in Java
- ◆ Tree interfaces in JDSL
 - InspectableBinaryTree
 - InspectableTree
 - BinaryTree
 - Tree
- ◆ Inspectable versions of the interfaces do not have update methods
- ◆ Tree classes in JDSL
 - NodeBinaryTree
 - NodeTree

- ◆ JDSL was developed at Brown's Center for Geometric Computing
- ◆ See the JDSL documentation and tutorials at <http://jdsl.org>

