CSCI 136
Data Structures & Advanced Programming

Trees

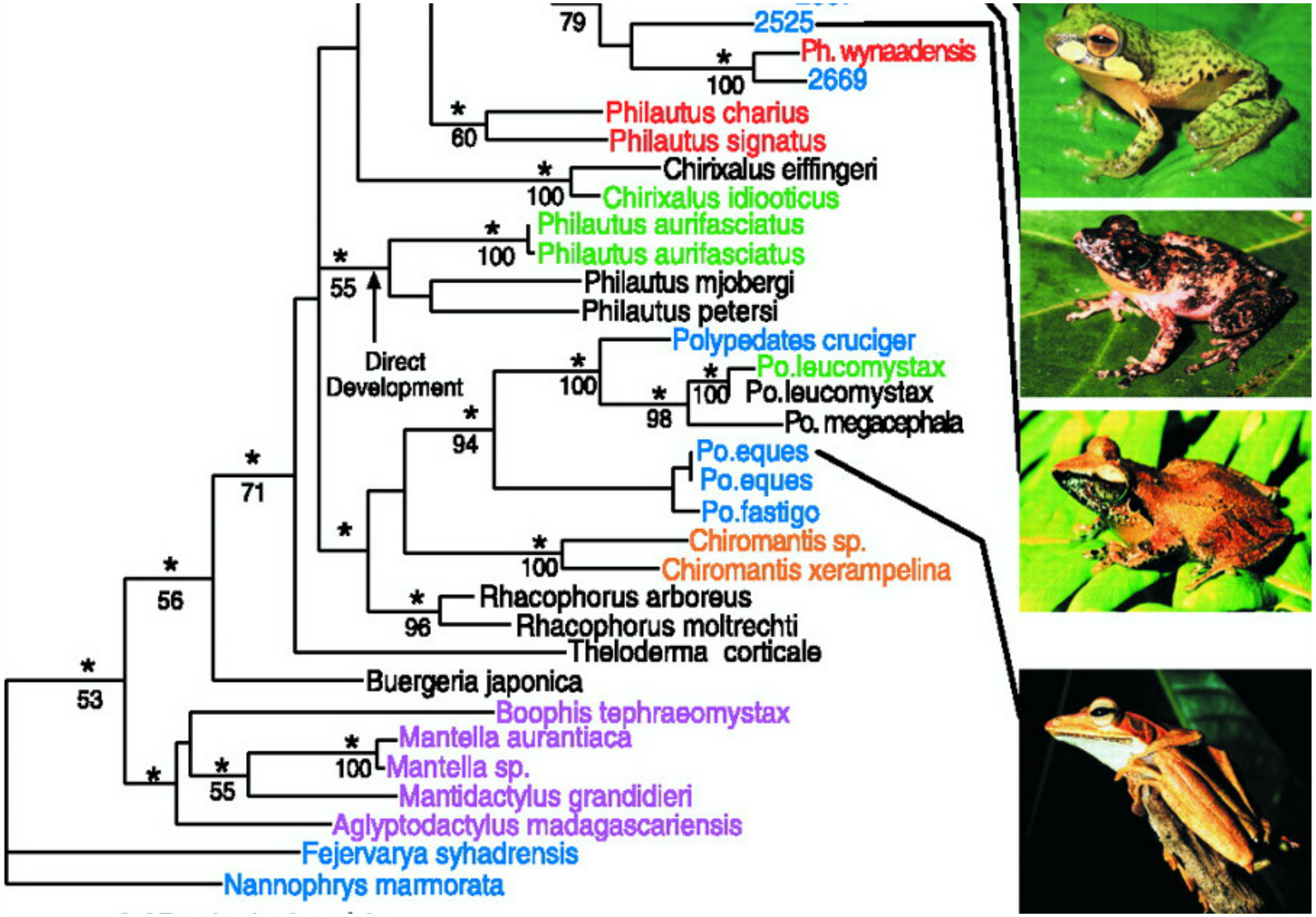
Trees

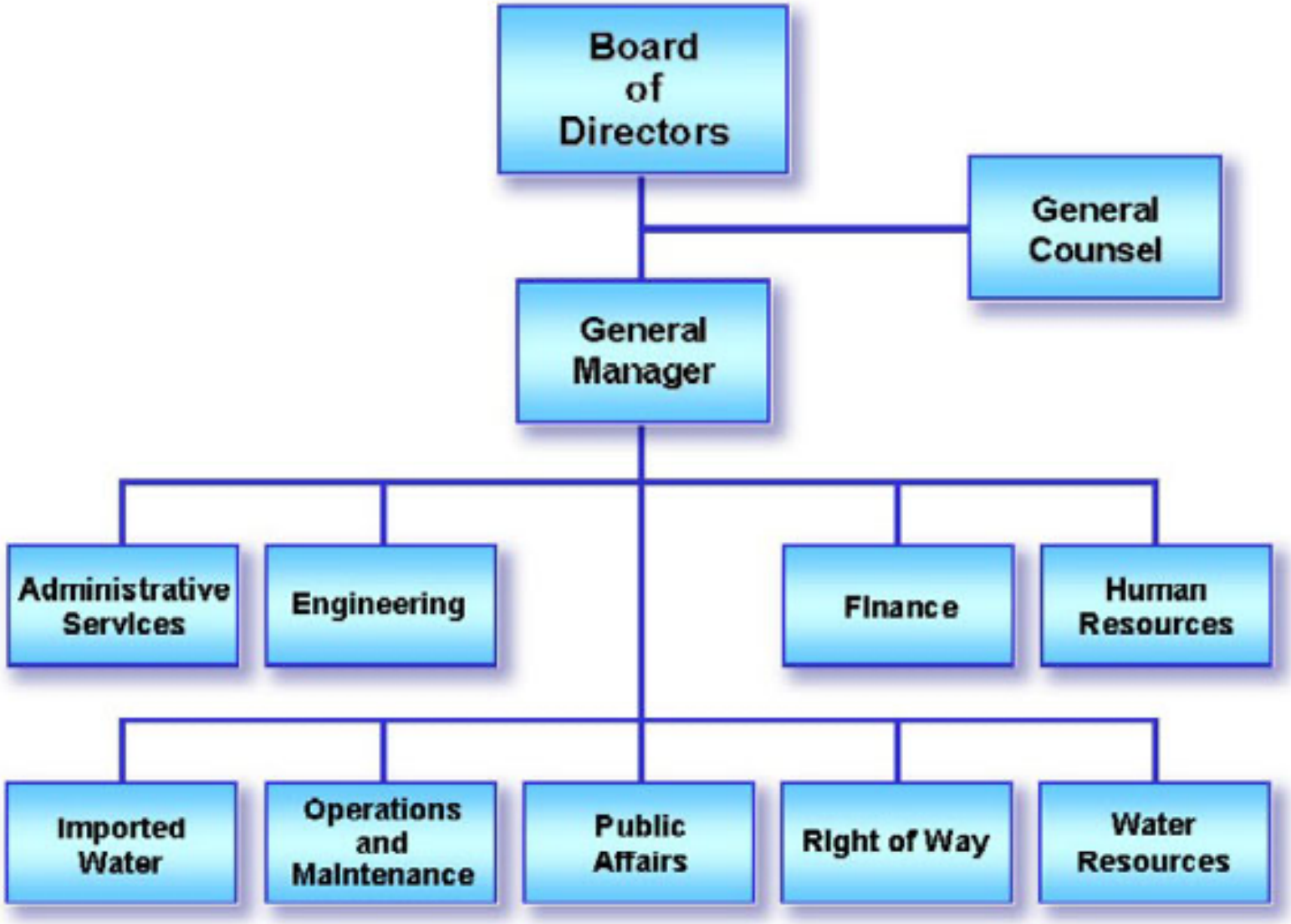
Overview of Tree Topics

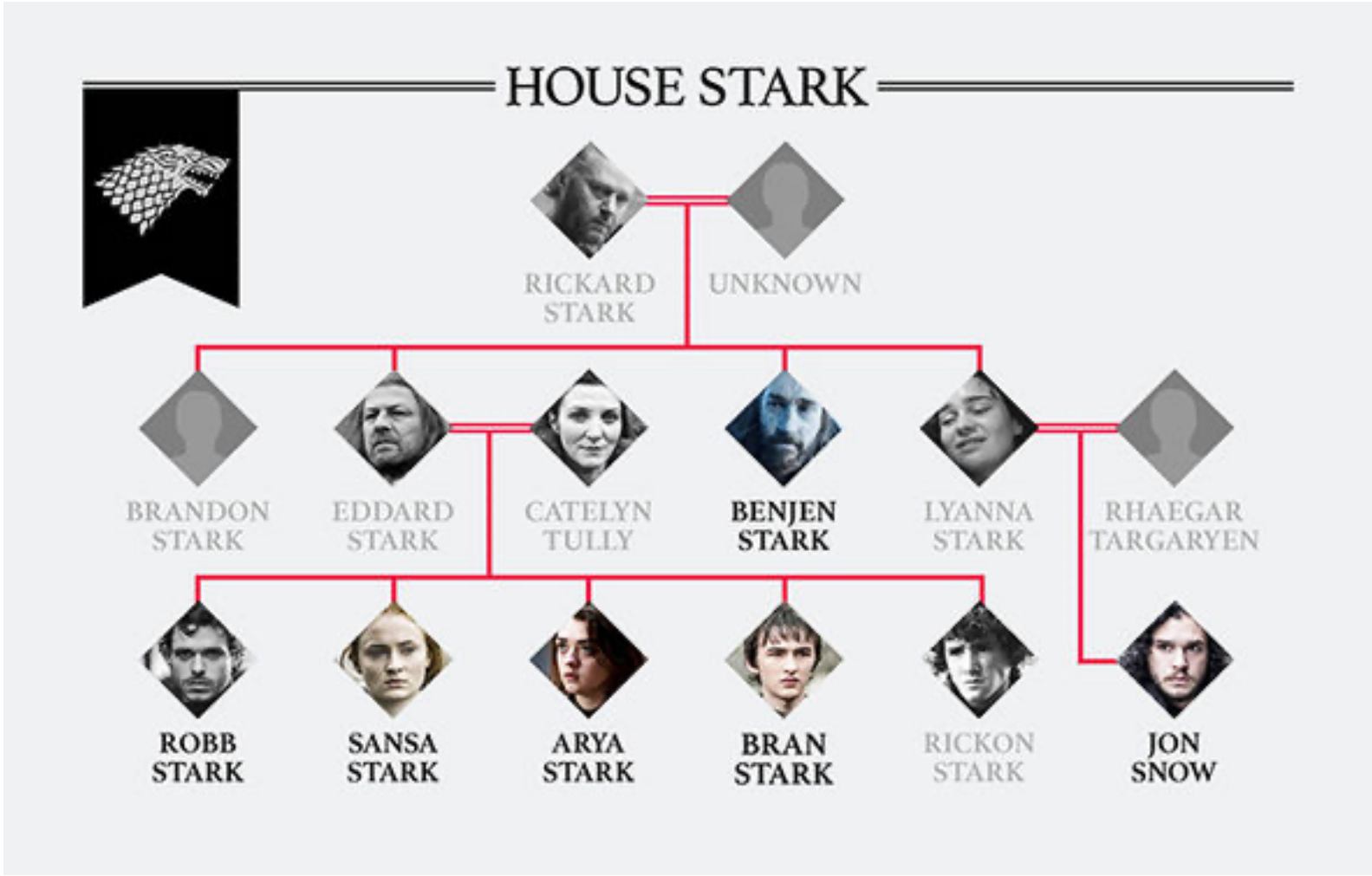
- Trees
  - Structure, Terminology, Examples
  - Implementation
  - Recursion/Induction on Trees
  - Applications
  - Traversals

This Presentation

- Trees
  - Structure, Terminology, Examples
  - Implementation

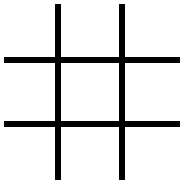

79
2525
*
Ph. wynaadensis
100
2669
*
Philautus charius
60
Philautus signatus
*
Chirixalus eiffingeri
100
Chirixalus idlootlcus
*
Philautus aurifasciatus
100
*
Philautus aurifasciatus
55
Philautus mjobergi
Philautus petersi
Direct
Development
Polypedates cruciger
*
Po.leucomystax
100
*
100
Po.leucomystax
98
Po. megacephala
*
94
Po.eques
*
71
Po.eques
Po.fastigo
*
Chiromantis sp.
*
100
Chiromantis xerampelina
*
56
*
Rhacophorus arboreus
96
Rhacophorus moltrechti
Theloderma corticale
*
53
Buergeria japonica
Boophis tephraeomystax
*
Mantella aurantiaca
100
Mantella sp.
*
*
55
Mantidactylus grandidieri
Aglyptodactylus madagascariensis
Fejervarya syhadrensis
Nannophrys marmorata


Board of Directors
General Counsel
General Manager
Administrative Services
Engineering
Finance
Human Resources
Imported Water
Operations and Maintenance
Public Affairs
Right of Way
Water Resources



express.co.uk

~lenhart

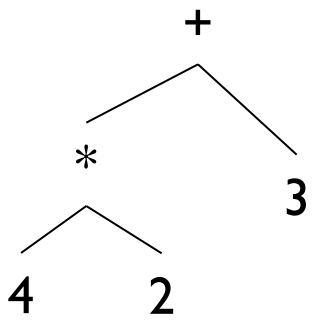


Expression Trees

NATALIE JEREMIJENKO: TREE LOGIC
(MassMoCA.org)

Expression Trees

4 * 2 + 3



Tree Features

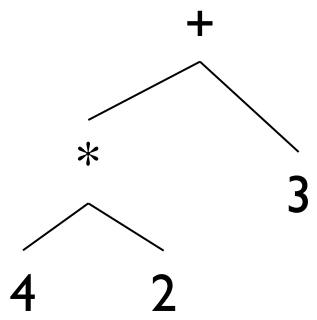
- Hierarchical relationship among nodes
- Root at the top
- Leaves at the bottom
- Interior nodes in middle
- Parents, children, ancestors, descendants, siblings
- Degree (of node): number of children of node
- Degree (of tree): maximum degree (across all nodes)
- Depth of node: number of *edges* from root to node
- Height of tree: maximum depth (across all nodes)

Introducing Binary Trees

- Degree of each node at most 2
- Recursive nature of tree
  - Empty
  - Root with left and right subtrees
- SLL: Recursive nature was captured by hidden node (Node<E>) class
- Binary Tree: No "inner" node class
  - Single BinaryTree class does it all
  - Is it a tree or a node?
    - It's a node that's a root of a tree!
  - And it's not part of Structure hierarchy!

Expression Trees

4 * 2 + 3



Build using constructor

```
new BinaryTree<E>(value, leftSubTree, rightSubTree)
```

```
BinaryTree<String> fourTimesTwo =  new BinaryTree<String>
        (“*”,new BinaryTree<String>(“4”),new BinaryTree<String>(“2”));

BinaryTree<String> fourTimesTwoPlusThree = new BinaryTree<String>
        (“+”, fourTimesTwo,  new BinaryTree<String>(“3”));
```

Expression Trees

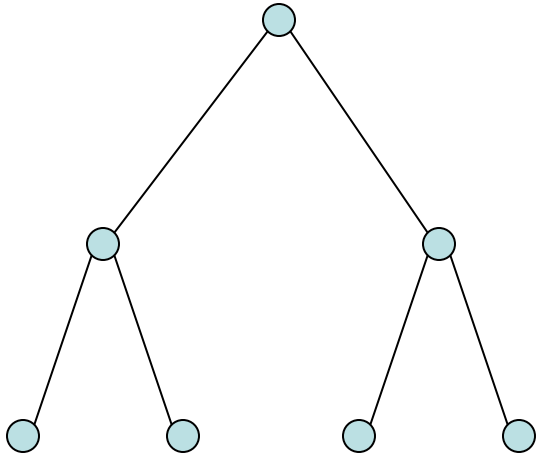
- General strategy
  - Make a binary tree (BT) for each leaf node
  - Move from bottom to top, creating BTs
  - Eventually reach the root
  - Call "evaluate" on final BT

- Example
  - How do we make a binary expression tree for (((4+3)*(10-5))/2)
    - Postfix notation: 4 3 + 10 5 - * 2 /
    - Think PostScript!
      - 4 3 add 10 5 sub mul 2 div

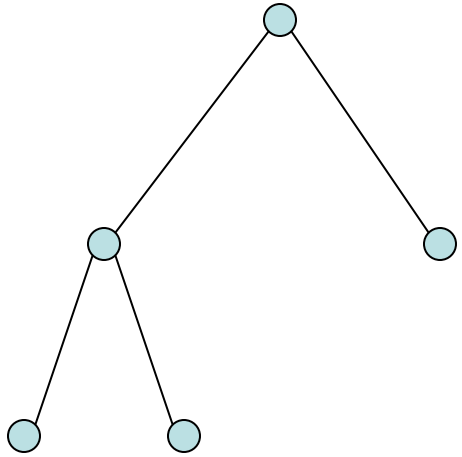
```
int evaluate(BinaryTree<String> expr) {


    if (expr.height() == 0)
      return Integer.parseInt(expr.value());


    else {
      int left = evaluate(expr.left());
      int right = evaluate(expr.right());
      String op = expr.value();
      switch (op) {

      case "+" : return left + right;
      case "-" : return left - right;
      case "*" : return left * right;
      case "/" : return left / right;
      }

      Assert.fail("Bad op");
      return -1;
   }
}
```

Full vs. Complete (non-standard!)

- Full tree – A full binary tree of height h has *leaves only* on level h, and each internal node has exactly 2 children.



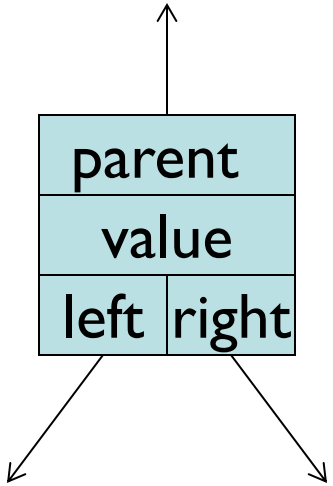
- Complete tree – A *complete* binary tree of height h is *full* to height h-1 and has all leaves at level h in leftmost locations.

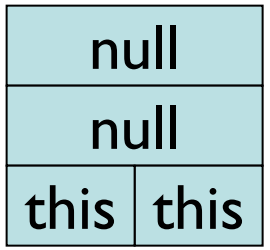


All full trees are complete, but not all complete trees are full!

Implementing BinaryTree

- BinaryTree<E> class
  - Instance variables
    - BinaryTree: parent, left, right
    - E: value
- left and right are never null
  - If no child, they point to an "empty" tree
    - Empty tree T has value null, parent null, left = right = T
  - Only empty tree nodes have null value

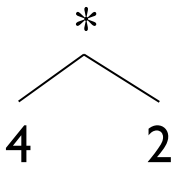


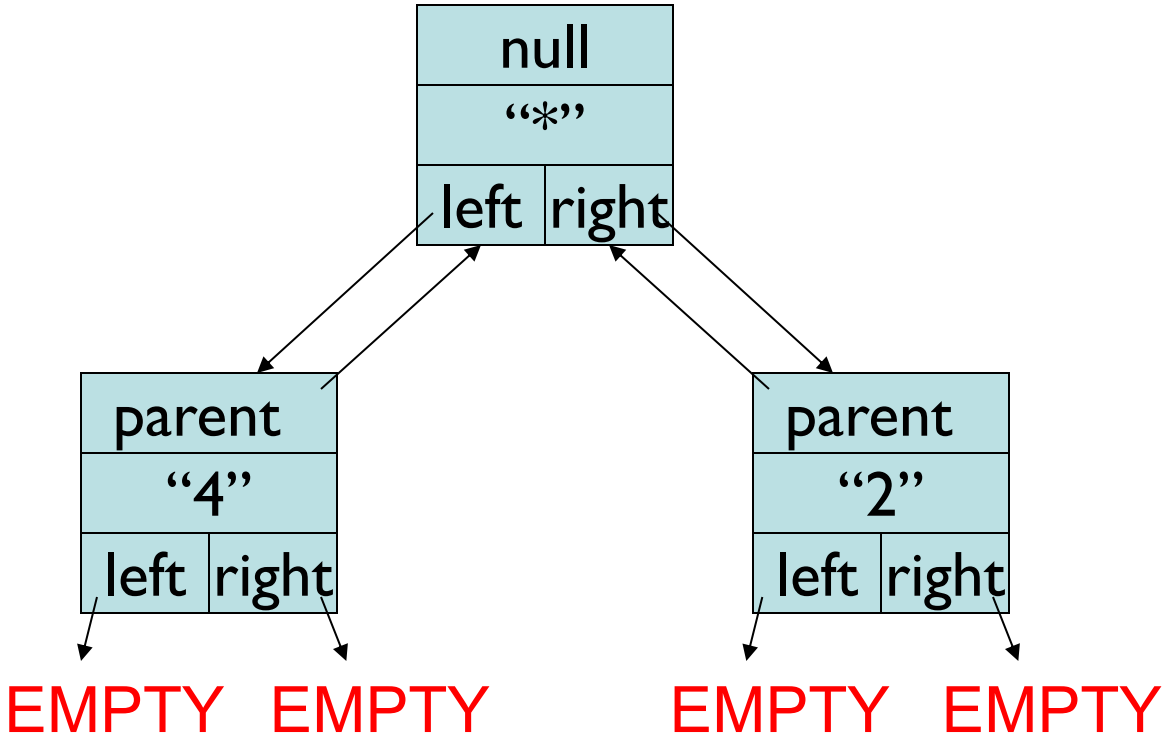


EMPTY BT

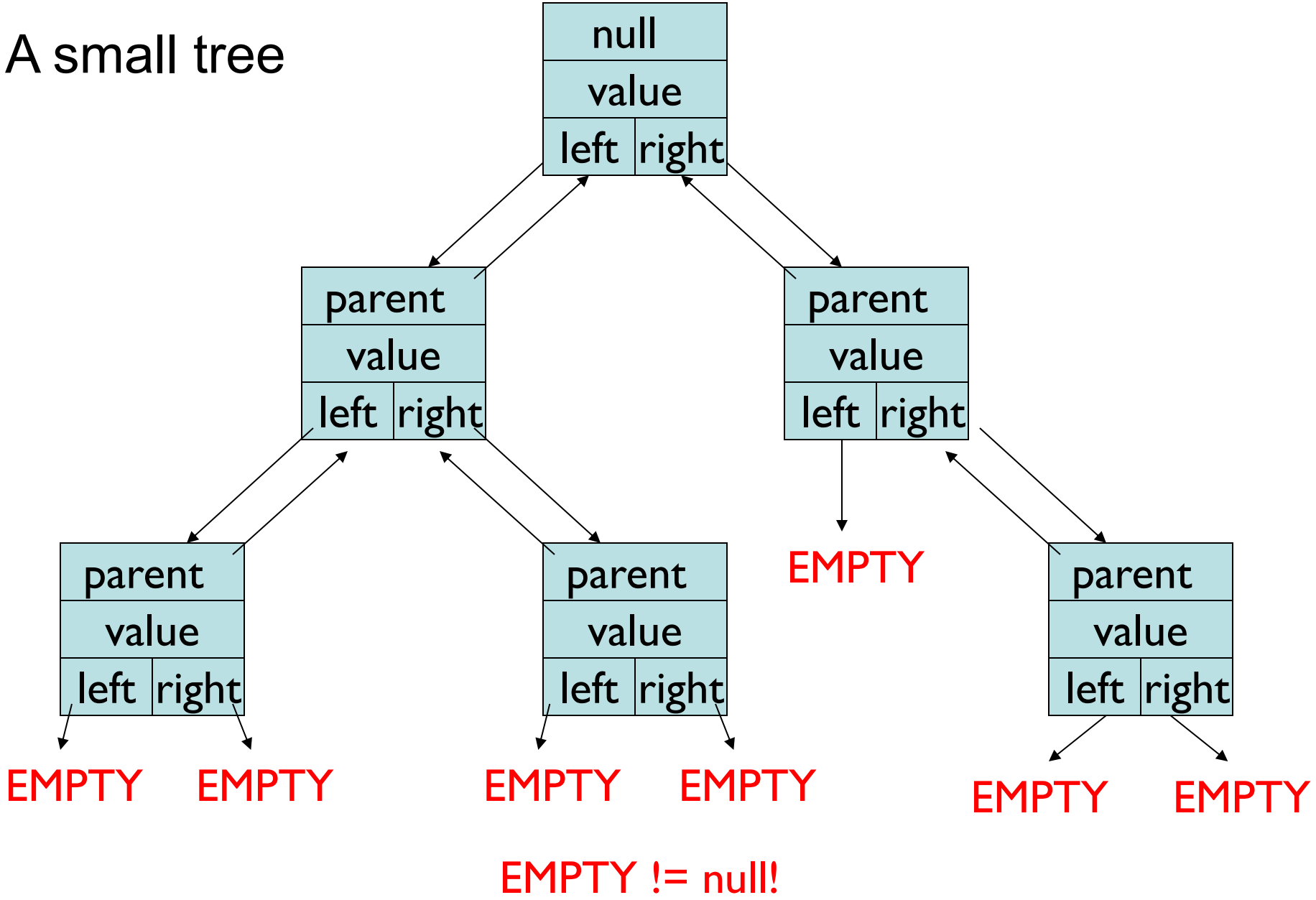
Implementing BinaryTree

- BinaryTree class
  - Instance variables
    - BT parent, BT left, BT right, E value





A small tree


null
value
left
right
parent
value
left
right
parent
value
left
right
parent
value
left
right
parent
value
left
right
parent
value
left
right
EMPTY
EMPTY
EMPTY
EMPTY
EMPTY
EMPTY
EMPTY
EMPTY
EMPTY != null!

Implementing BinaryTree

- Many (!) methods: See BinaryTree javadoc page
- All "left" methods have equivalent "right" methods
  - public BinaryTree()
    - // generates an empty node (EMPTY)
    - // parent and value are null, left=right=this
  - public BinaryTree(E value)
    - // generates a tree with a non-null value and two empty (EMPTY) subtrees
  - public BinaryTree(E value, BinaryTree<E> left, BinaryTree<E> right)
    - // returns a tree with a non-null value and two subtrees
  - public void setLeft(BinaryTree<E> newLeft)
    - // sets left subtree to newLeft
    - // re-parents newLeft by calling newLeft.setParent(this)
  - protected void setParent(BinaryTree<E> newParent)
    - // sets parent subtree to newParent
    - // called from setLeft and setRight to keep all "links" consistent

Implementing BinaryTree

- Methods:
  - public BinaryTree<E> left()
    - // returns left subtree
  - public BinaryTree<E> parent()
    - // post: returns reference to parent node, or null
  - public boolean isLeftChild()
    - // returns true if this is a left child of parent
  - public E value()
    - // returns value associated with this node
  - public void setValue(E value)
    - // sets the value associated with this node
  - public int size()
    - // returns number of (non-empty) nodes in tree
  - public int height()
    - // returns height of tree rooted at this node
  - But where's "remove" or "add"?!?!

Three BinaryTree Constructors

```
public class BinaryTree<E> {
   protected E val; // value associated with node
   protected BinaryTree<E> parent; // parent of node
   protected BinaryTree<E> left, right; // children

   // Construct an empty tree
   public BinaryTree() {
      val = null;
      parent = null; left = right = this;
   }
```

Three BinaryTree Constructors

```
// Construct a single-node tree (a leaf)
public BinaryTree(E value) {
    Assert.pre(value != null, "must be non-null.");
    val = value;
    right = left = new BinaryTree<E>();
    setLeft(left);
    setRight(right);
}
```

Three BinaryTree Constructors

```
// Construct a single-node tree (a leaf)
public BinaryTree(E value, BinaryTree<E> left,
                           BinaryTree<E> right) {
   Assert.pre(value != null, "must be non-null.");
   val = value;
   if (left == null) { left = new BinaryTree<E>(); }
   setLeft(left);
   if (right == null) { right = new BinaryTree<E>(); }
   setRight(right);
}
```

Tree Surgery

```
// Replace left subtree of this node
// There's also 'setRight' and 'setParent'
public void setLeft(BinaryTree<E> newLeft) {
    if (isEmpty()) return;
    // cut off left sub-tree and delete parent ref
    if (left != null && left.parent() == this)
        left.setParent(null);
    left = newLeft;
    left.setParent(this);
}
```

Many Recursive Methods

```
public int size() {
    if (isEmpty()) return 0;
    return left().size() + right().size() + 1;
}
public BinaryTree<E> root() {
    if (parent() == null) return this;
    else return parent().root();
}
public int height() {
    if (isEmpty()) return -1;
    return 1 + Math.max(left.height(),right.height());
}
```

Implementation Notes

- Many methods are recursive
  - Read through the class implementation: BinaryTree<E>
  - Understanding recursion as applied to trees will reinforce your skills at recursion
  - Later on, we'll discuss the isBalanced method and the iterator-producing methods
- Adjusting parent reference is a protected method
  - Why?
    - For a node to reset its parent, it needs to know which child of its parent it should become!
    - Compare with: For a node to reset one of its children, it's clear who the parent of the new child is!

Summary & Observations

- Trees can encode
  - Hierarchical relationships
  - Decision procedures
  - Evolutionary/Genealogical relationships
  - Organizational structures
  - Evolving system/game state
- Trees are naturally recursive structures
- BinaryTree class implements a tree data structure in which a node has at most two children
  - In fact, it represents a single node that we interpret as the root of a possibly larger binary tree.
  - Unlike the linkedList structures, the node is exposed.
    - The allow users to write recursive methods for trees