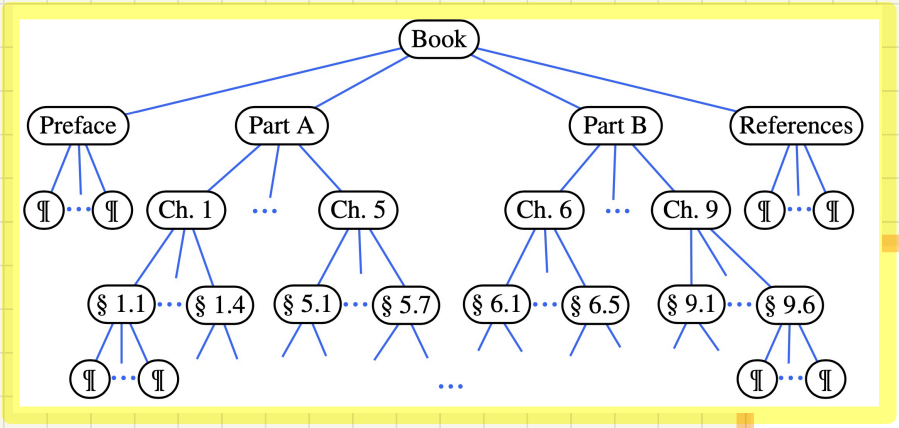
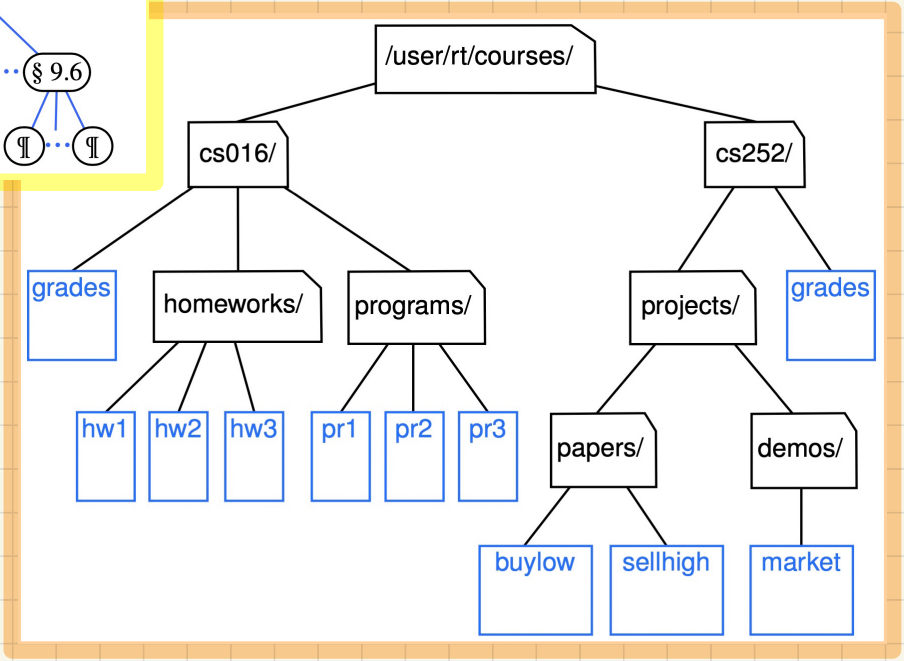
# General Trees: Recursive Definition

- root
- size

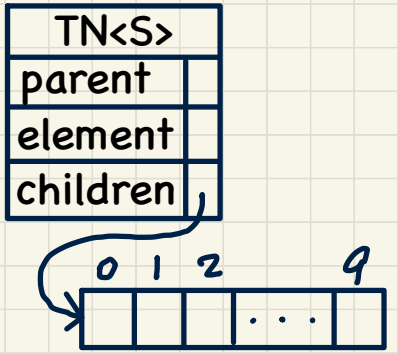
# General Trees: Ordered vs. Unordered Trees



# Generic, General Tree Nodes

```
public class TreeNode<E> {
  private E element; /* data object */
  private TreeNode<E> parent; /* unique parent node */
  private TreeNode<E>[] children; /* list of child nodes */

  private final int MAX_NUM_CHILDREN = 10; /* fixed max */
  private int noc; /* number of child nodes */

  public TreeNode(E element) {
    this.element = element;
    this.parent = null;
    this.children = (TreeNode<E>[])
      Array.newInstance(this.getClass(), MAX_NUM_CHILDREN);
    this.noc = 0;
  }
  public E getElement() { ... }
  public TreeNode<E> getParent() { ... }
  public TreeNode<E>[] getChildren() { ... }

  public void setElement(E element) { ... }
  public void setParent(TreeNode<E> parent) { ... }
  public void addChild(TreeNode<E> child) { ... }
  public void removeChildAt(int i) { ... }
}
```




**Compare**:

+ prev ref.
+ next ref.

in a DLN.

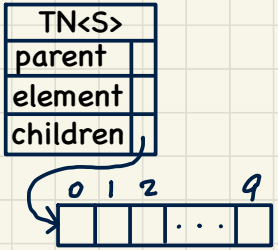
# Instantiating Generic Structures in Java

```
class ArrayStack<E> {
  private E[] data;
  …
  public ArrayStack<E>() {
    
  }
}
```

```
class TreeNode<E> {
  private TreeNode<E>[] children;
  …
  public TreeNode<E>() {
    
  }
}
```

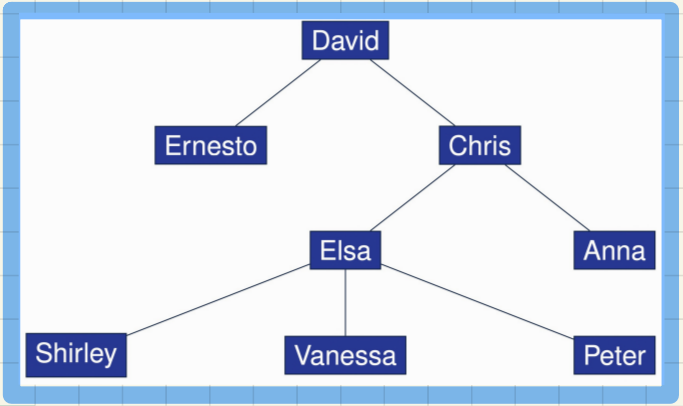
# Tracing: Constructing a Tree

```java
@Test
public void test_general_trees_construction() {
  TreeNode<String> agnarr = new TreeNode<>("Agnarr");
  TreeNode<String> elsa = new TreeNode<>("Elsa");
  TreeNode<String> anna = new TreeNode<>("Anna");

  agnarr.addChild(elsa);
  agnarr.addChild(anna);
  elsa.setParent(agnarr);
  anna.setParent(agnarr);

  assertNull(agnarr.getParent());
  assertTrue(agnarr == elsa.getParent());
  assertTrue(agnarr == anna.getParent());
  assertTrue(agnarr.getChildren().length == 2);
  assertTrue(agnarr.getChildren()[0] == elsa);
  assertTrue(agnarr.getChildren()[1] == anna);
}
```



# Tracing: Computing a Node's Depth

```
public int depth(TreeNode<E> n) {
  if(n.getParent() == null) {
    return 0;
  }
  else {
    return 1 + depth(n.getParent());
  }
}
```




depth(vanessa)

```
@Test
public void test_general_trees_depths() {
  ... /* constructing a tree as shown above */
  TreeUtilities<String> u = new TreeUtilities<>();
  assertEquals(0, u.depth(david));
  assertEquals(1, u.depth(ernesto));
  assertEquals(1, u.depth(chris));
  assertEquals(2, u.depth(elsa));
  assertEquals(2, u.depth(anna));
  assertEquals(3, u.depth(shirley));
  assertEquals(3, u.depth(vanessa));
  assertEquals(3, u.depth(peter));
}
```

# Binary Trees: Recursive Definition

- root
- size

# Deriving the Sum of a Geometric Sequence

Initial Term: I

Common Factor: r

Number of Terms: k