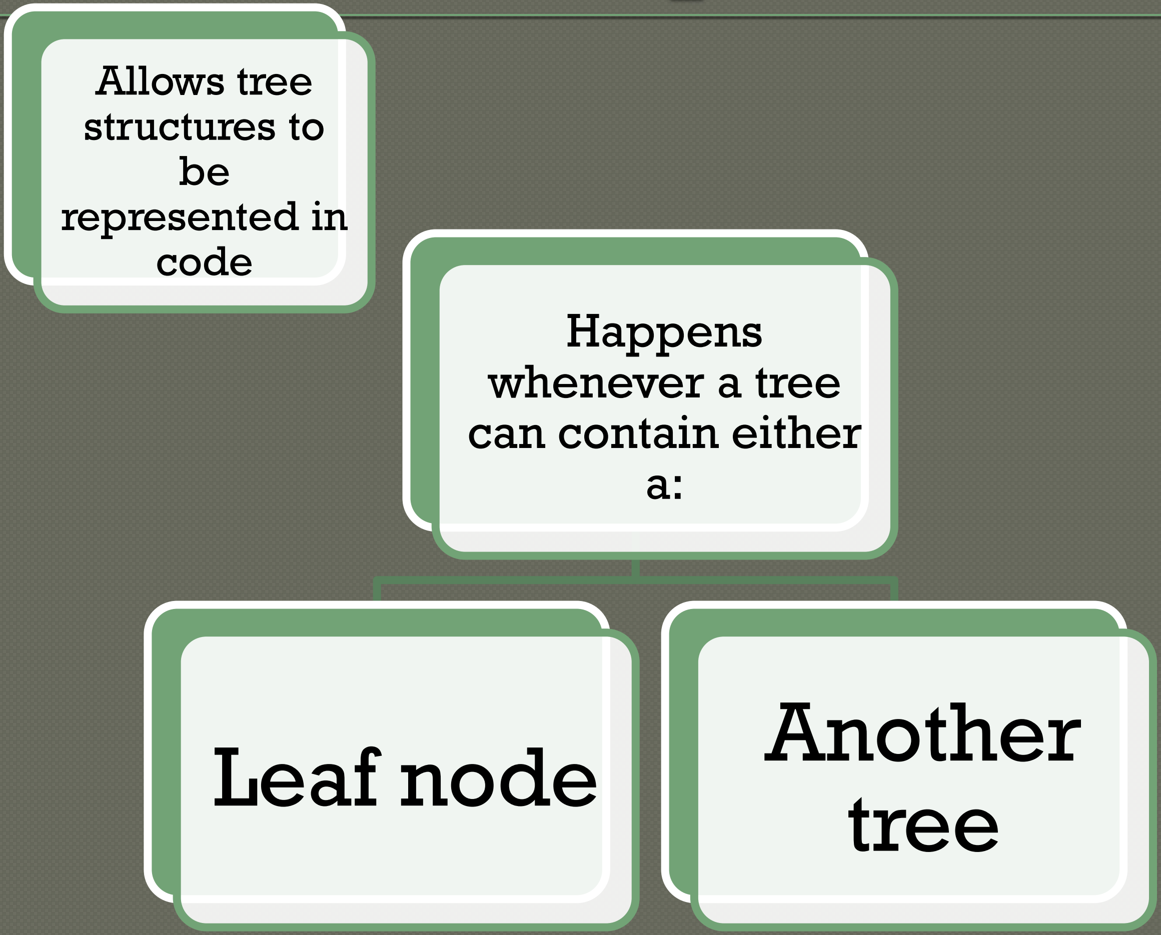
Design Patterns

- The Strategy Pattern
- The Factory Method
- Generics
- The Abstract Factory Pattern
- The State Pattern
- The Observer Pattern
- The Adapter Pattern
- **The Composite Pattern**
- The Iterator Pattern
- The Builder Pattern
- Fallen Patterns
  - The Singleton Pattern
  - The Visitor Pattern

The Composite Pattern

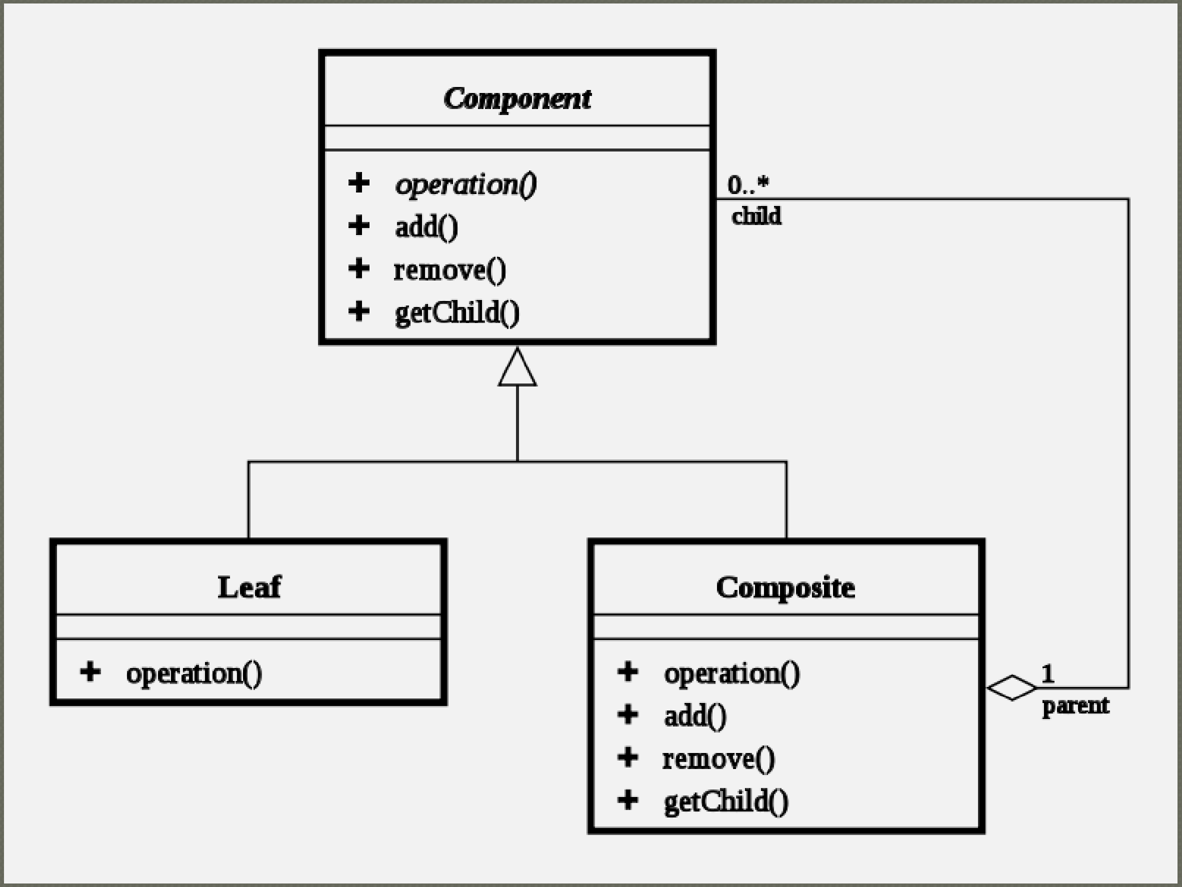


Allows tree structures to be represented in code

Happens whenever a tree can contain either a:

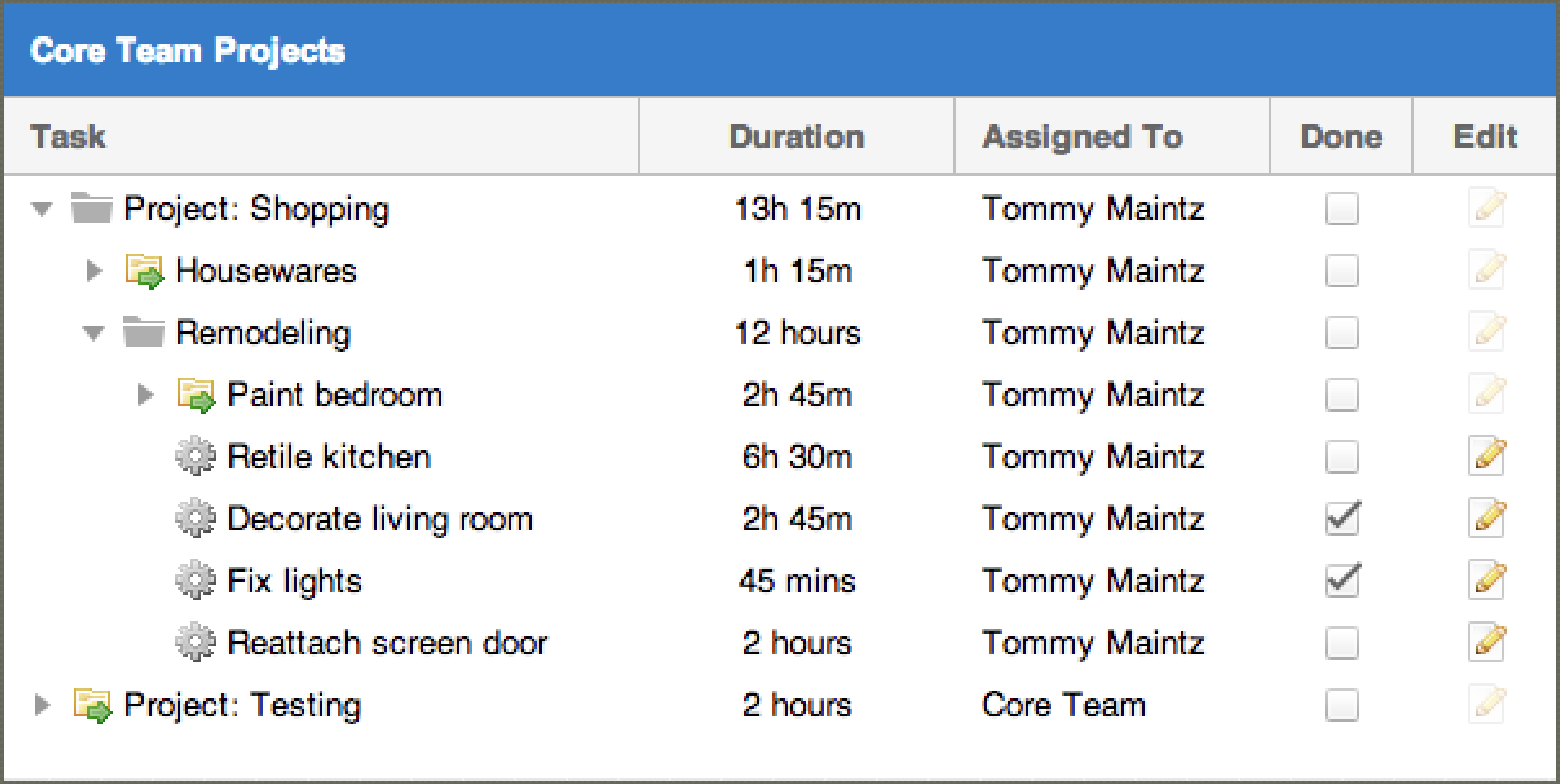
- Leaf node
- Another tree

Example Structure



https://codeblitz.wordpress.com/2009/07/29/perfect-match-composite-and-visitor-pattern/

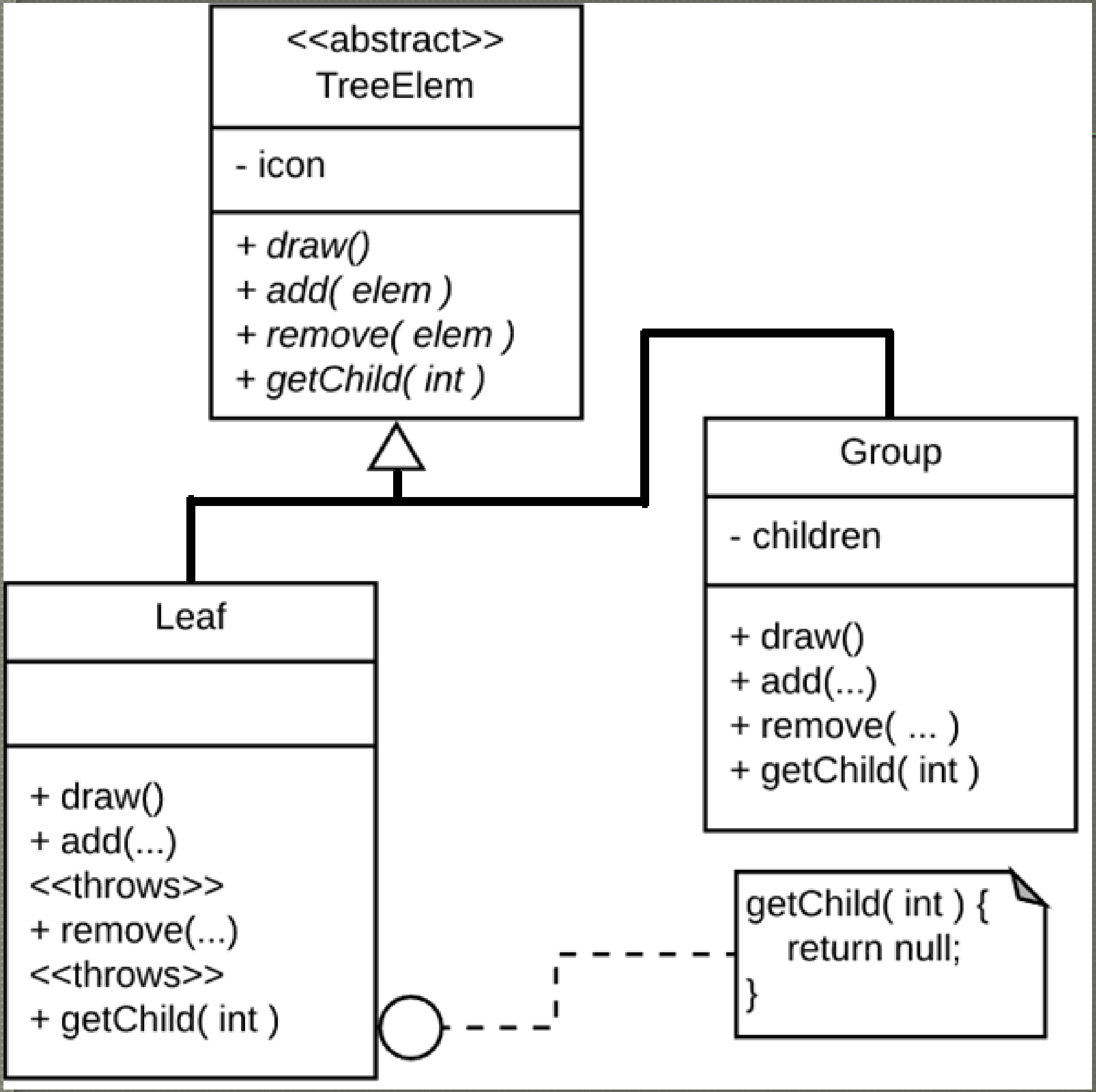
Simple Example: Tree View



Core Team Projects

Task	Duration	Assigned To	Done	Edit
Project: Shopping	13h 15m	Tommy Maintz	☐	✎
Housewares	1h 15m	Tommy Maintz	☐	✎
Remodeling	12 hours	Tommy Maintz	☐	✎
Paint bedroom	2h 45m	Tommy Maintz	☐	✎
Retile kitchen	6h 30m	Tommy Maintz	☐	✎
Decorate living room	2h 45m	Tommy Maintz	✓	✎
Fix lights	45 mins	Tommy Maintz	✓	✎
Reattach screen door	2 hours	Tommy Maintz	☐	✎
Project: Testing	2 hours	Core Team	☐	✎

What do Tree Elements Look Like?



```
getChild( int ) {
    return null;
}
```

- draw() is implemented by both leaves and groups
- Add/remove is an error for leaves (this may be bad)

Code for TreeElem

```
//Note: we’re omitting “parent”,
//but it’s allowed

public abstract class TreeElem {
    private Graphic icon;

    public abstract void draw();
    public abstract void add( TreeElem child );
    public abstract void remove(TreeElem chld);
    public abstract TreeElem getChild( int );
    public TreeElem( Graphic icon ) {
        this.icon = icon;
    }
}
```

Leaf is Easy

```
public class Leaf extends TreeElem {
    public void draw() { … }

    public void add( … ) { throw … }
    public void remove( … ) { throw … }
    public TreeElem getChild( int ) {
        return null;
    }
    public TreeElem( Graphic icon ) {
        super(icon);
    }
}
```

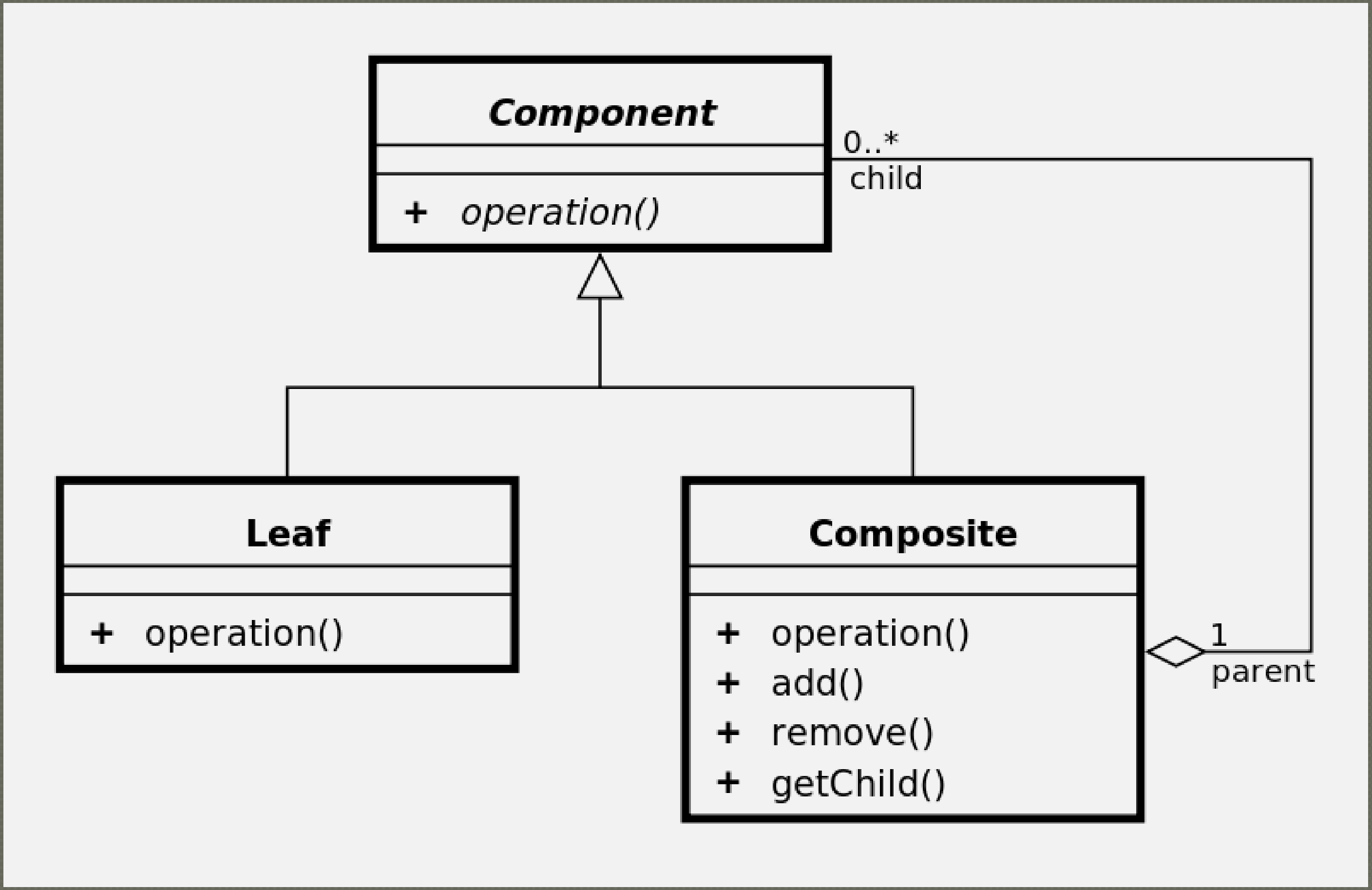
Group is More Involved

```
public class Group extends TreeElem {
    private List<TreeElem> children;

    public void draw() {
        super.draw();
        for( TreeElem child : children() )
            { child.draw(); }
    }
    public void add( TreeElem child )
        { children.add( child ); }
    public void remove( TreeElem child ) {…}
    public TreeElem getChild( int i )
        { return children.get(i); }

    public Group(Graphic icon) {
        super( icon );
        children = new HashSet<TreeElem>();
    }
}
```

Problems with design…

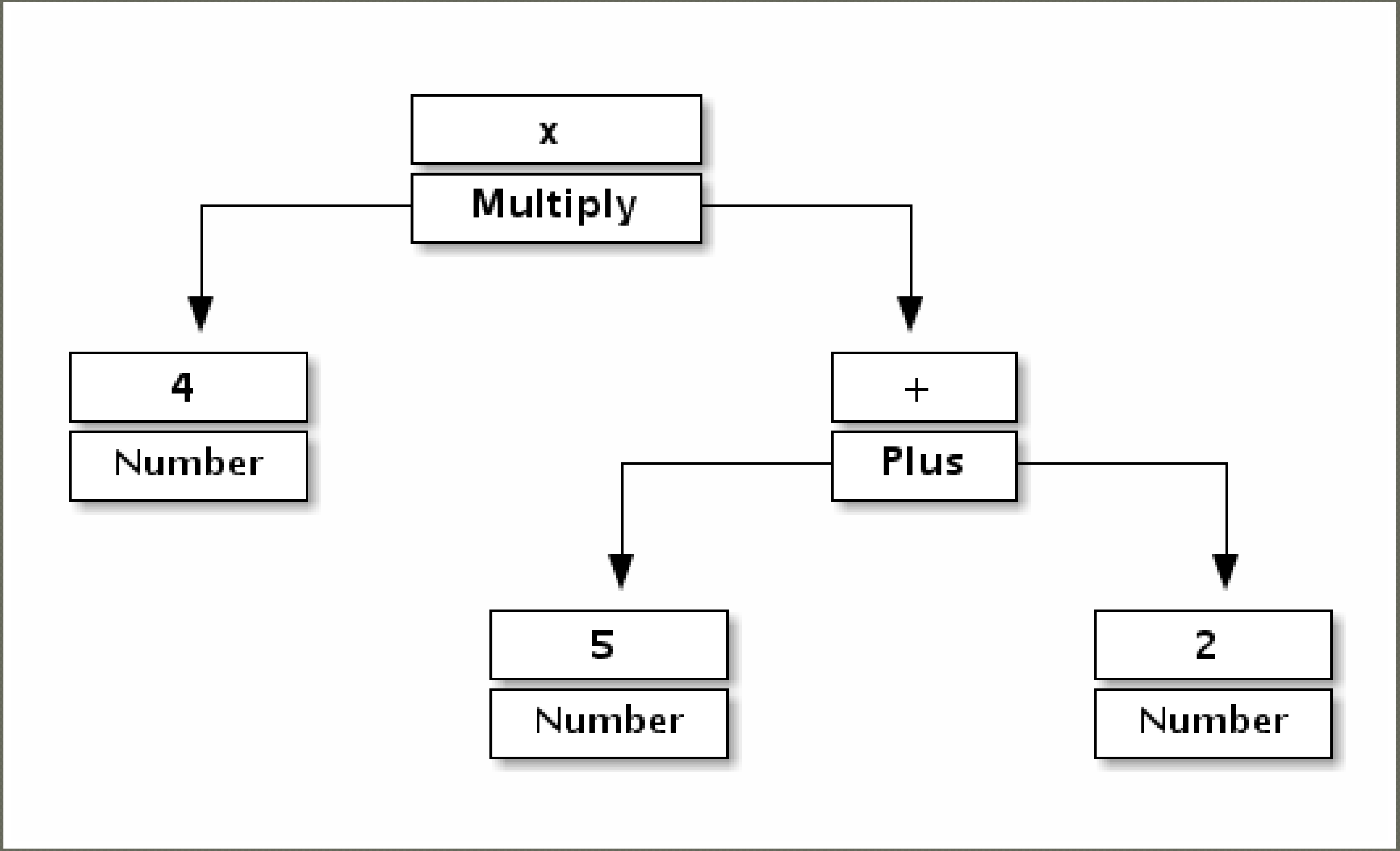
- Ideas?
- Think about SOLID principles
- Main Issues
  - Add/remove really shouldn't be in the base class
  - Why does leaf do anything?
  - How do we decide how much state to "leak" from the composite itself?
- Give it some thought…

Better Composite



https://www.revolvy.com/topic/Composite%20pattern

Example: Abstract Syntax Trees



Example AST for the expression: "4 x 5 + 2"

What are these for?

- When you parse an expression, it needs to be turned into a data structure
- A tree is the most natural structure for most languages
- Expressions must be nestable:
  - `X + ( Y * ( A + B ) )`
- The objects in a tree must be self-similar

New Principle

- Some designers believe in the “Law of Demeter”: http://wiki.c2.com/?LawOfDemeter
- “Classes should only call their own methods or fields’ methods.”
  - Fields means member variables here
- “They should never call/access two levels deep”
- Example:
  - Ok: foo.doBaz(); //doBaz() calls bar.baz()
  - Bad: foo.getBar().baz();

Apply Composite