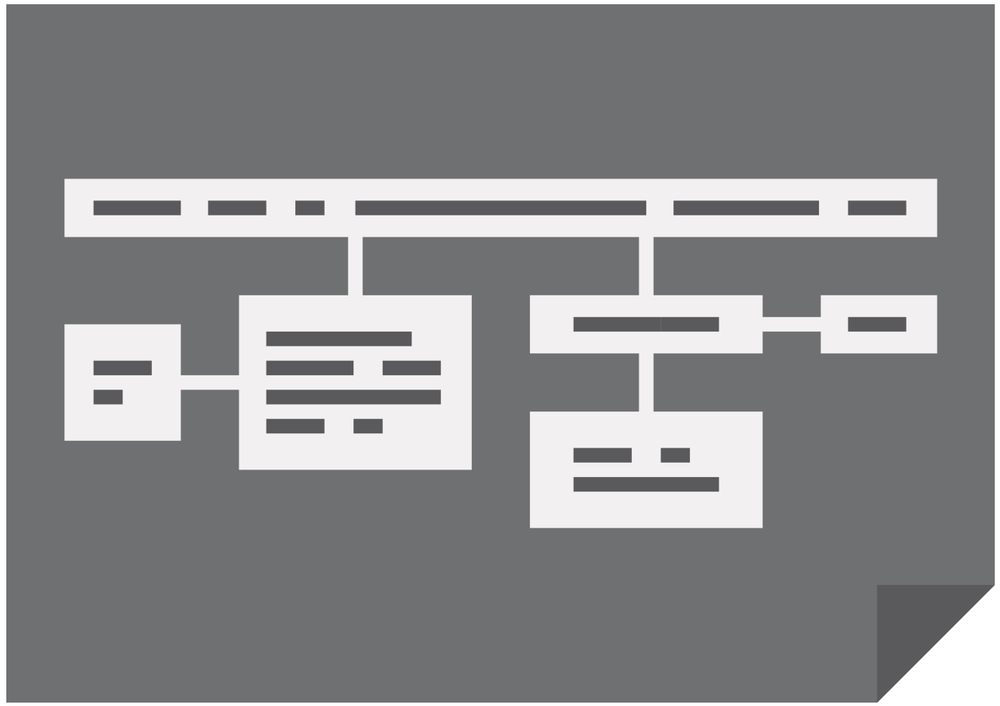
Composite Pattern

Bryan Hansen

twitter: bh5k | http://www.linkedin.com/in/hansenbryan

Concepts

- Components represent part or whole structure
- Compose objects into tree structures
- Individual object treated as a Composite
- Same operations applied on individual and composites
- Examples:
  - java.awt.Component
  - JSF widgets
  - RESTful service GETs



Design

Tree structured

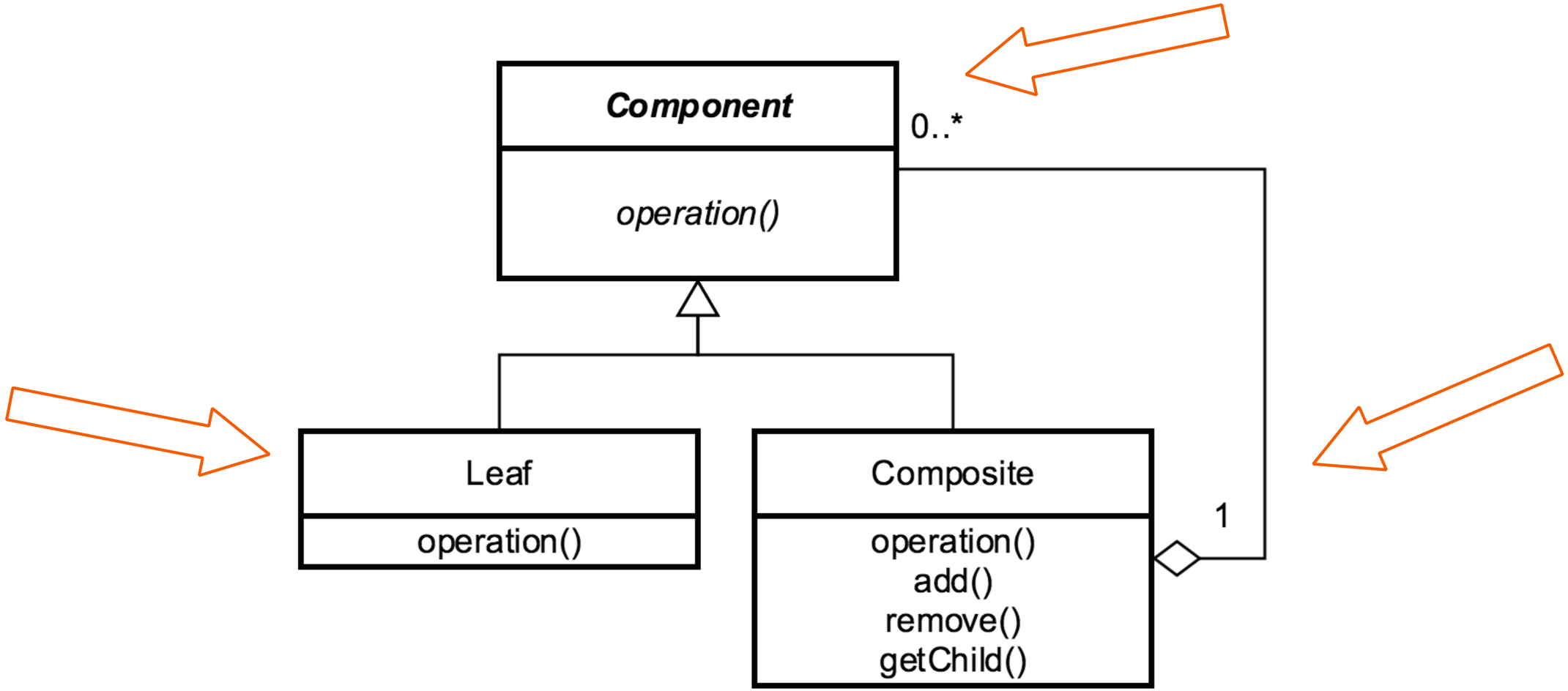
Component

Leaf or Composite, same operations

Composite knows about child objects

Component, Leaf, Composite

UML


Component
operation()
0..*
Leaf
operation()
Composite
operation()
add()
remove()
getChild()
1

Everyday Example - Map

```
Map<String, String> personAttributes = new HashMap<>();

personAttributes.put("site_role", “person");
personAttributes.put("access_role", "limited");

Map<String, String> groupAttributes = new HashMap<>();

groupAttributes.put("group_role", "claims");

Map<String, String> secAttributes = new HashMap<>();

secAttributes.putAll(personAttributes);
secAttributes.putAll(groupAttributes);
```

Exercise Composite

Menu, MenuItem, MenuComponent

Create Composite

Features Not Supported

Pitfalls

- Can overly simplify system
- Difficult to restrict
- Implementation can possibly be costly

Contrast

Composite

- Tree structure
- Leaf and Composite have same interface
- Unity between objects

Decorator

- Contains another entity
- Modifies behavior (adds)
- Doesn’t change underlying object

Composite Summary

- Generalizes a hierarchical structure
- Can simplify things too much
- Easier for clients
- Composite != Composition