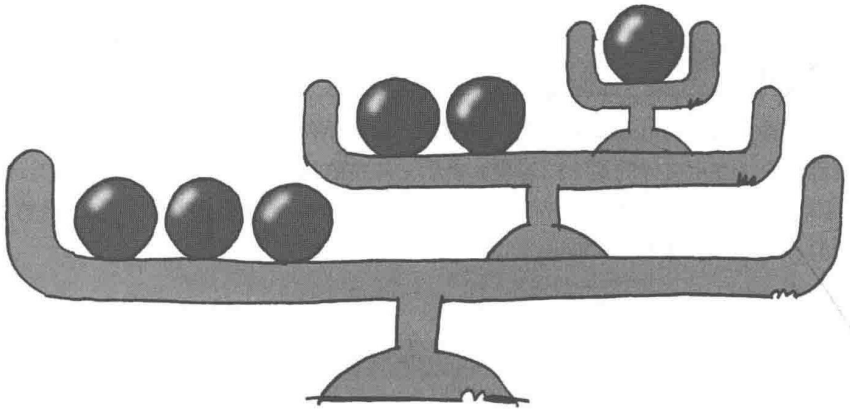
Composite (组合, Structural Pattern)



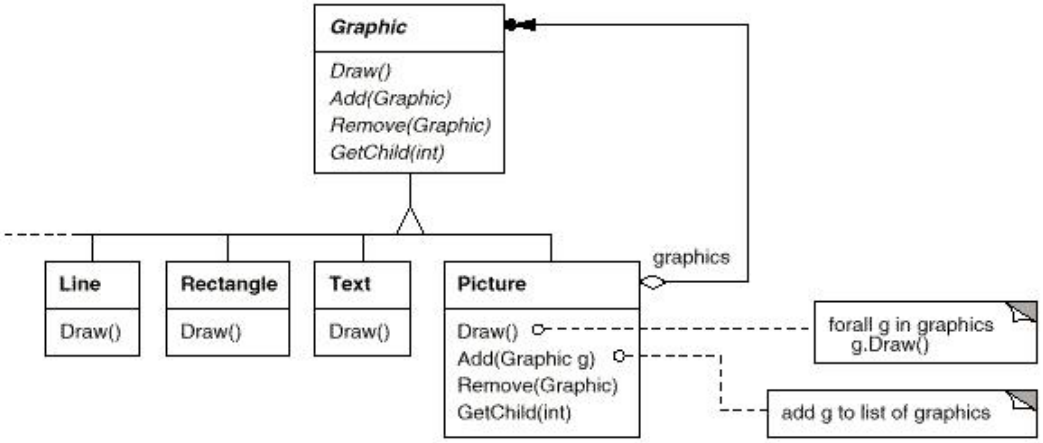
Kai SHI

Intent

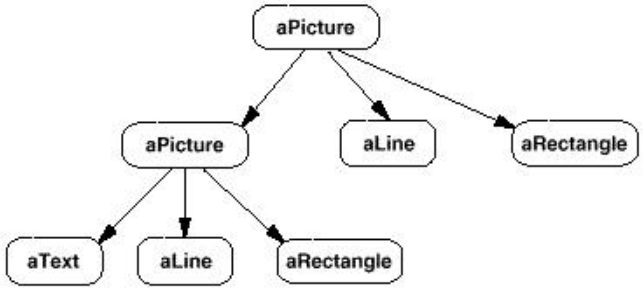
- Compose objects into tree structures to represent part-whole hierarchies.
- Composite lets clients treat individual objects and compositions of objects uniformly.
- Similar to Generalized List (广义表) in data structure.

Motivation

- Graphics applications like drawing editors and schematic capture systems let users build complex diagrams out of simple components. The user can group components to form larger components, which in turn can be grouped to form still larger components.



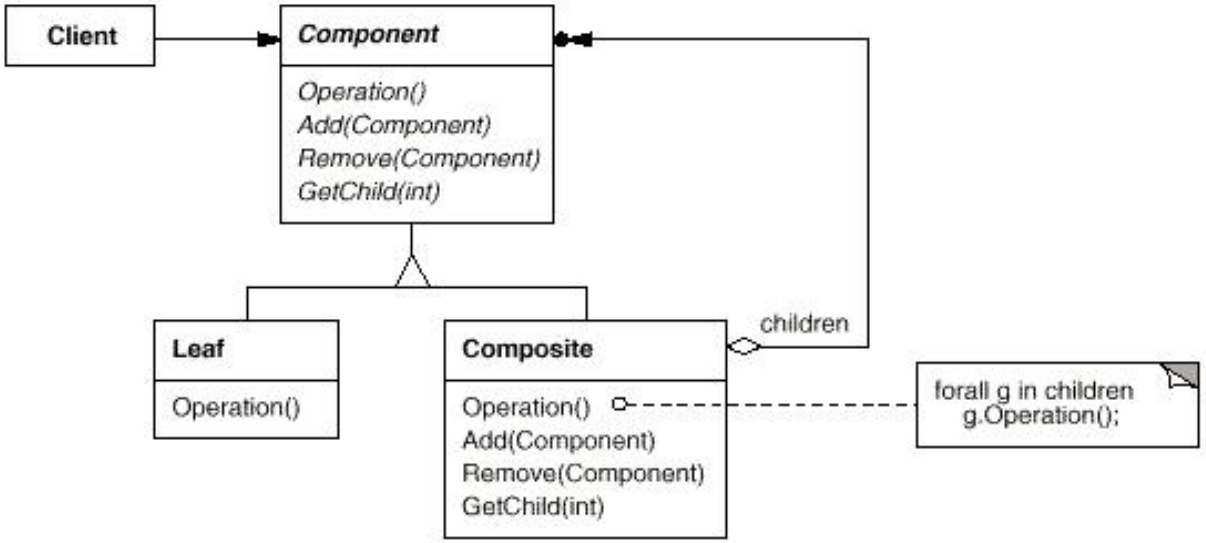
- The following diagram shows a typical composite object structure of recursively composed Graphic objects:



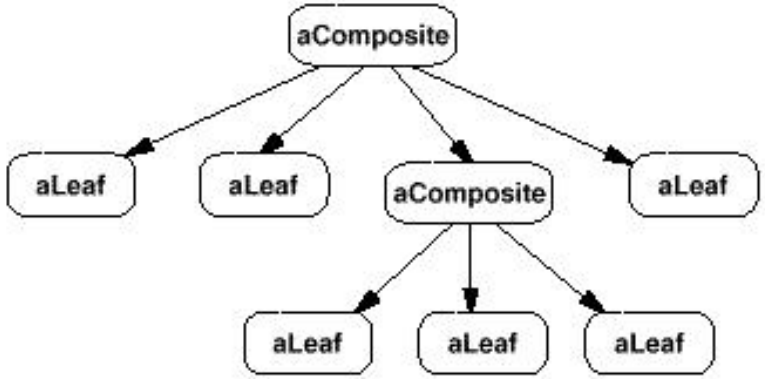
Applicability:
Use the Composite pattern when

- you want to represent part-whole hierarchies of objects.
- you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.

Structure



- A typical Composite object structure might look like this:



Participants

- Component
  - Declares the interface for objects in the composition.
  - Implements default behavior for the interface common to all classes, as appropriate.
  - Declares an interface for accessing and managing its child components.
  - (Optional) Defines an interface for accessing a component's parent in the recursive structure, and implements it if that's appropriate.
- Leaf
  - Represents leaf objects in the composition. A leaf has no children.
  - Defines behavior for primitive objects in the composition.
- Composite
  - Defines behavior for components having children.
  - Stores child components.
  - Implements child-related operations in the Component interface.
- Client
  - Manipulates objects in the composition through the Component interface.

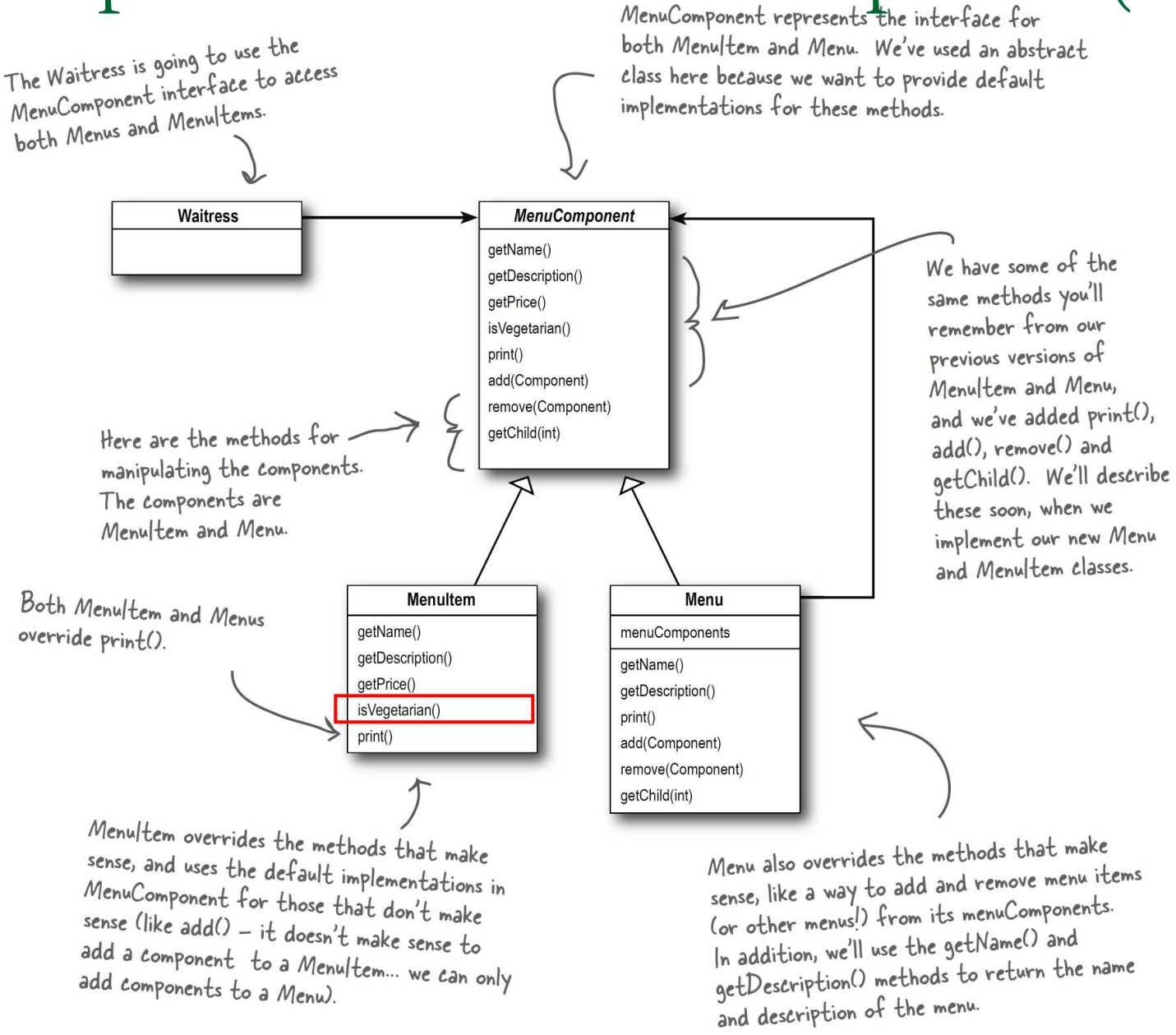
Collaborations

- Clients use the Component class interface to interact with objects in the composite structure.
  - If the recipient is a Leaf, then the request is handled directly.
  - If the recipient is a Composite, then it usually forwards requests to its child components, possibly performing additional operations before and/or after forwarding.

Consequences: The Composite pattern

- defines class hierarchies consisting of primitive objects and composite objects.
- makes the client simple.
  - Clients can treat composite structures and individual objects uniformly.
- makes it easier to add new kinds of components.
- can make the design overly (过于) general.
  - The disadvantage of making it easy to add new components is that it makes it harder to restrict the components of a composite.

Example: Menus with Composite (1/2)



Example: Menus with Composite (2/2)

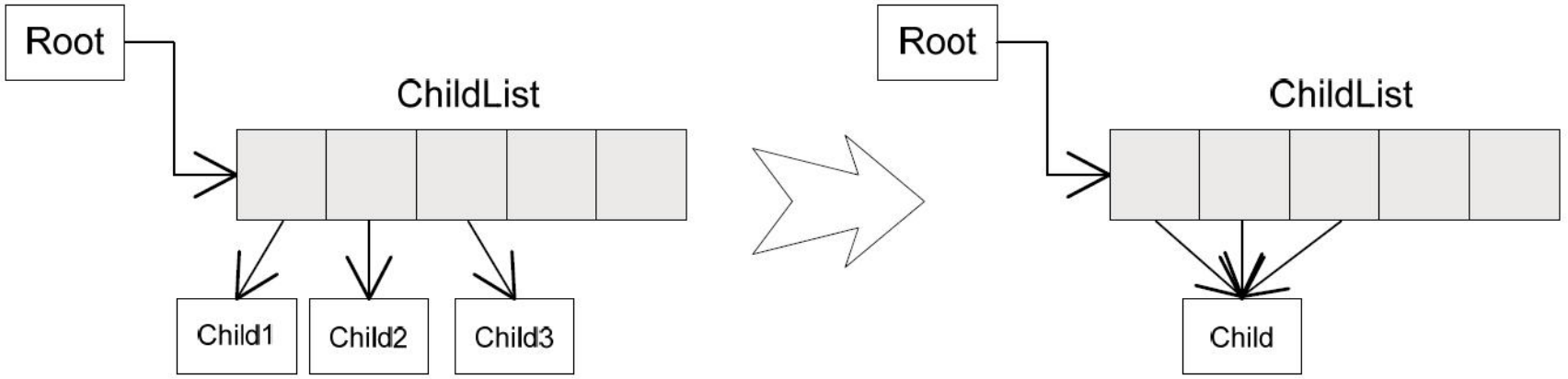
- Code:
  net.dp.composite.menu.MenuTestDrive

Implementation 1: Explicit parent references (bi-direction reference)

- Similar with Trifurcate Linked List (三叉链表)
- Maintaining references from child components to their parent can simplify the traversal and management of a composite structure.
  - The usual place to define the parent reference is in the Component class. Leaf and Composite classes can inherit the reference and the operations that manage it.
- It is unnecessary to let clients maintain bi-directions. Usually parent-to-children references are maintained by clients, child-toparent reference are maintained inside composite pattern automatically.

Implementation 2: Sharing components

- It's often useful to share components, for example, to reduce storage.
  - The component must be stateless or sharable state.



Implementation 3: Maximizing the Component interface

- Composite pattern makes clients unaware of the specific Leaf or Composite classes they're using.
  - Component class should define as many common operations for Composite and Leaf classes as possible.
  - There are many operations that Component supports that don't seem to make sense for Leaf classes. So that it conflict with Interface Segregation Principle (ISP) and Liskov Substitution Principle (LSP).
- How can Leaf provide a default implementation for them?
  - Make the useless operations, do nothing, or return null, or return mock object, or throws exception.

Implementation 4: Declaring the child management operations

- Should we declare "child management operations" in the Component and make them meaningful for Leaf classes, or should we declare and define them only in Composite and its subclasses?
- The decision involves a trade-off between safety and transparency:
  - Transparency: Defining the child management interface in the Component gives transparency, because you can treat all components uniformly. It costs you safety, because clients may try to add and remove objects from leaves.
  - Safety: Defining child management in the Composite class gives safety, but lose transparency, because leaves and composites have different interfaces.

Implementation 5: Child ordering

- Many designs specify an ordering on the children of Composite.
- When child ordering is an issue, you must design child access and management interfaces carefully to manage the sequence of children.

(有序树要注意子树的顺序)

Implementation 6: Caching to improve performance

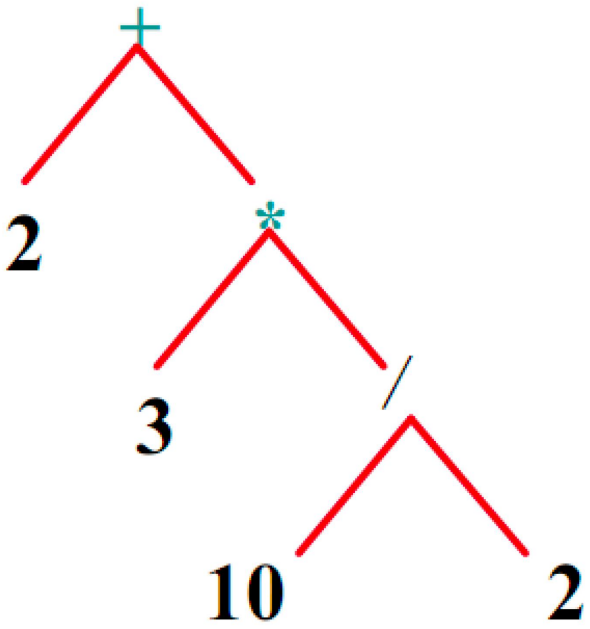
- If you need to traverse or search compositions frequently, the Composite class can cache traversal or search information about its children.
- Changes to a component will require invalidating (使无效) the caches of its parents.
  - This works best when components know their parents.
- So if you're using caching, you need to define an interface for telling composites that their caches are invalid.

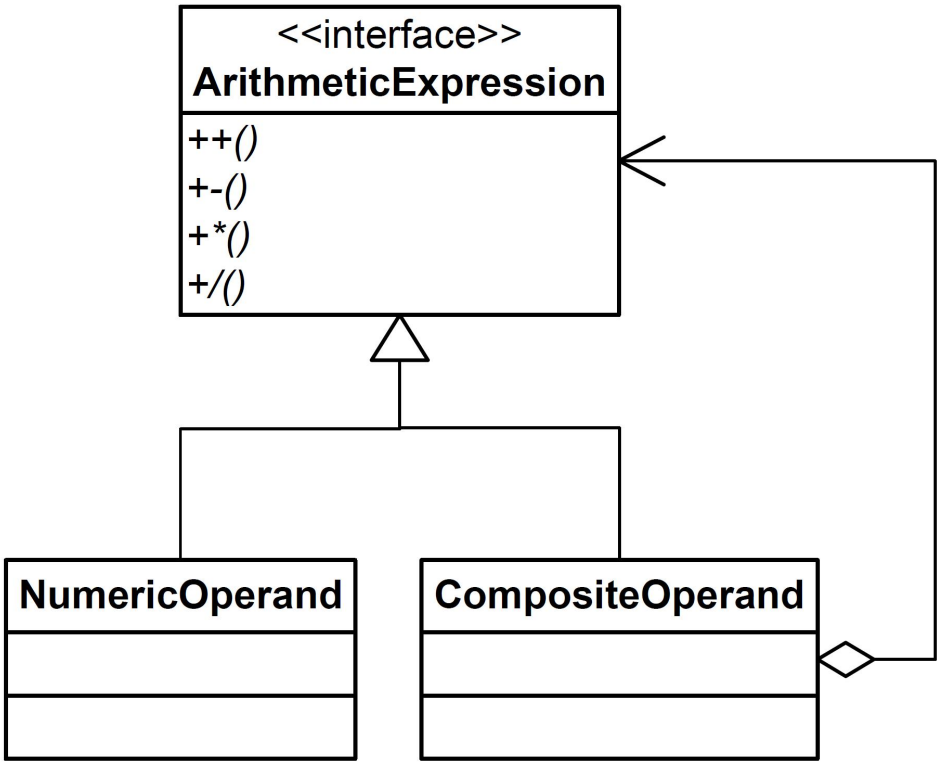
Implementation 7: What's the best data structure for storing components?

- Composites may use a variety of data structures to store their children,
  - Arrays, List, Set, HashMap
- The choice of data structure depends on efficiency.

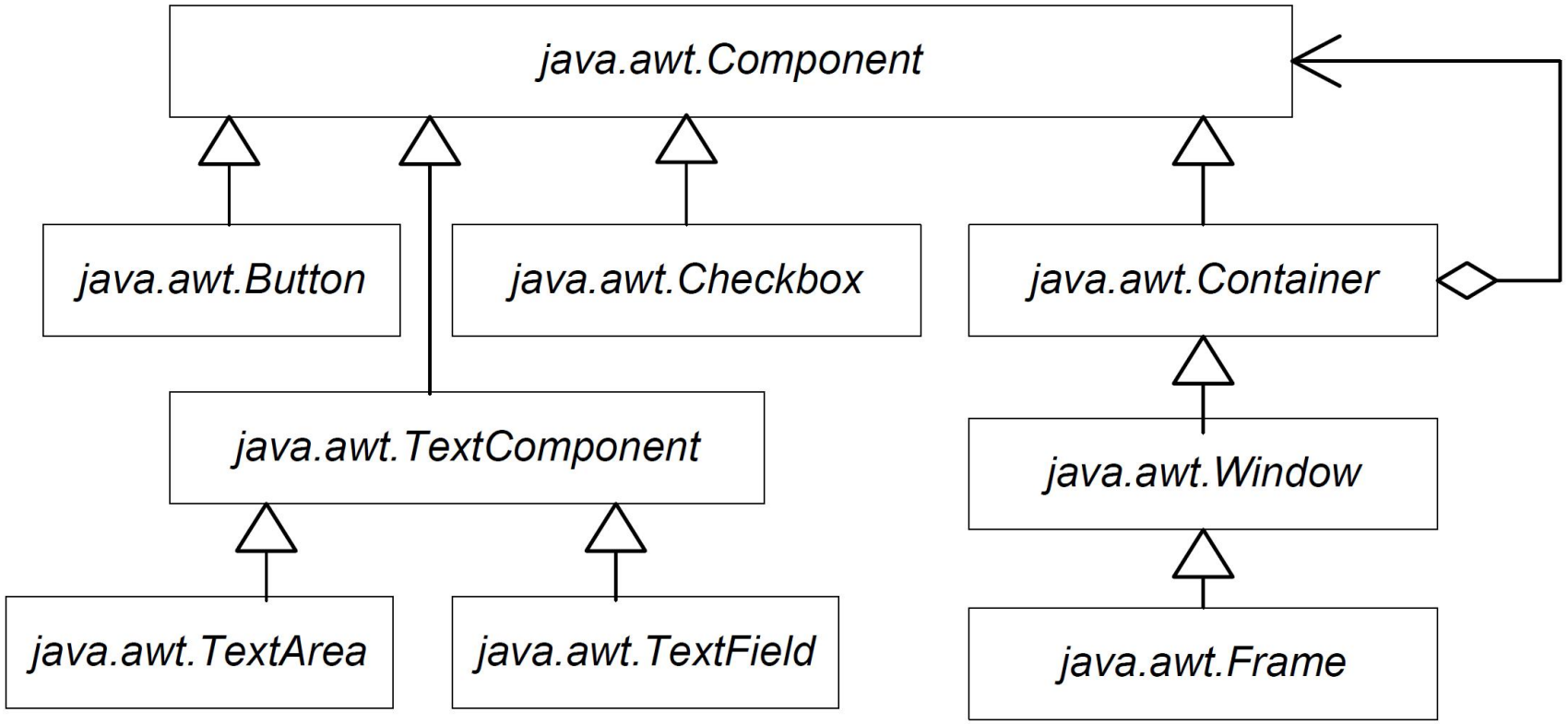
Example 1: Arithmetic expressions

- Arithmetic expressions can be expressed as trees where an operand can be a number or an arithmetic expression.

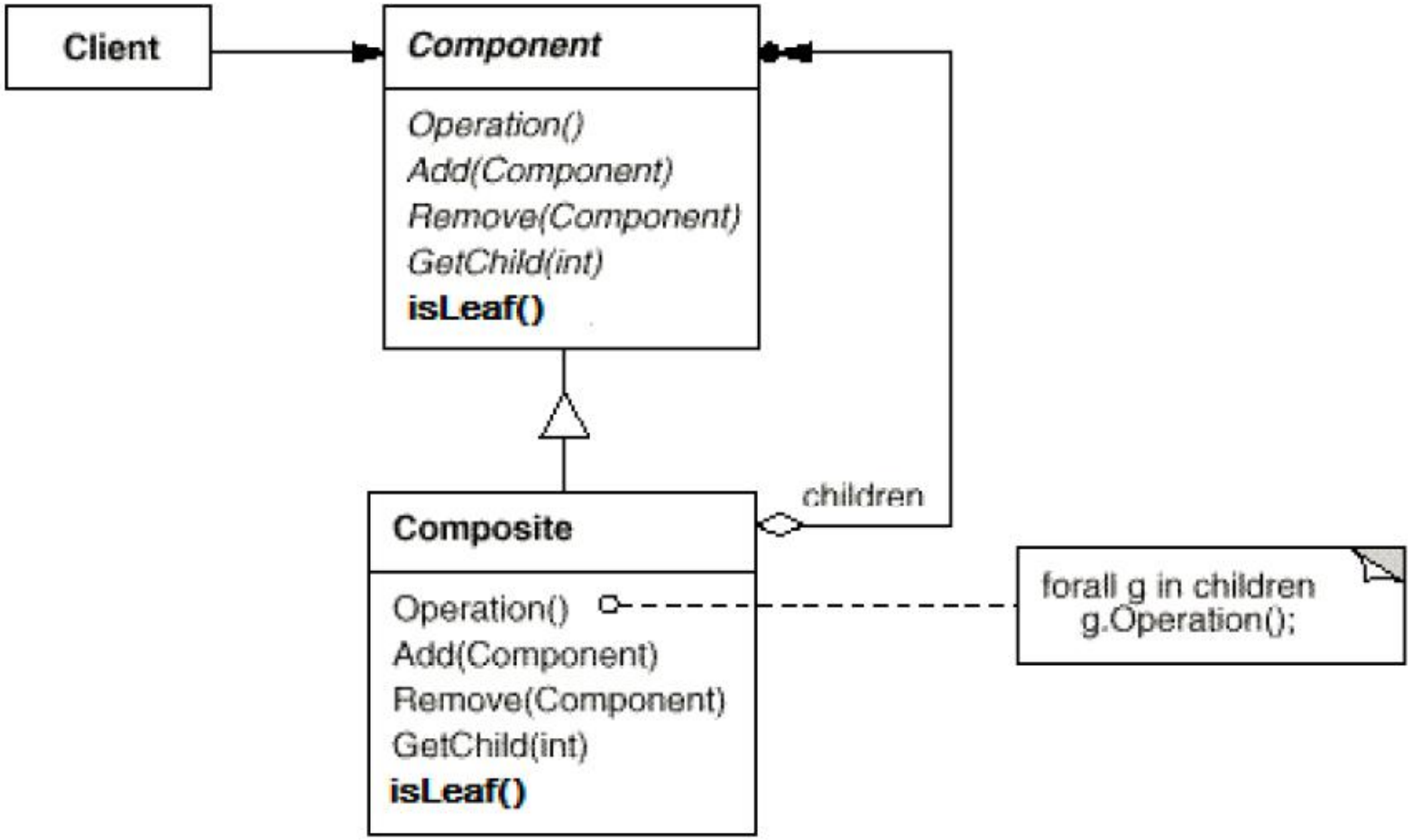




Example 2: Java AWT



Variation: Leaf and composite be one class



Thinking

- Requirements: print the vegetarian menu
  - Ans: net.dp.composite.menuiterator.MenuTestDrive