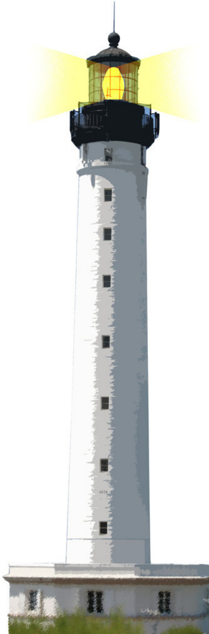
Advanced Object-Oriented Design

Composite

A nice and common design pattern

S. Ducasse

http://www.pharo.org



Outline

- Motivating examples
- Composite design pattern presentation
- Composite discussions

File entry examples

```
Pharo.image
```

```
F1
  Pharo.image
  Pharo.changes
```

```
F1
  src
  doc
  images
    Pharo.image
    Pharo.changes
```

File entries

An entry is a

- file
- or a folder with entries as children

Same with Trees

A tree is a

- leave
- or a node with trees as children

Documents

A document is composed of

- title
- table of contents
- chapters

A chaper is composed of sections
A section is composed of

- paragraphs
- figures
- lists
- sections

A diagram

- A diagram is composed of elements
- An element is
  - a circle
  - a segment
  - a text
  - a group (i.e, diagram)

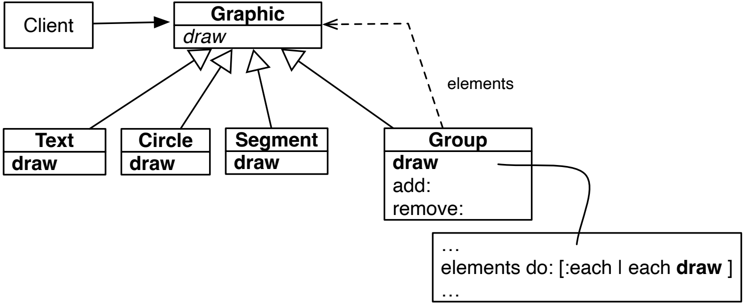
Now the question!

- How do we draw diagram elements?
- How do we draw a diagram?

We do not want to have to check if we are talking to an element or a diagram composed of elements!

Composite motivation

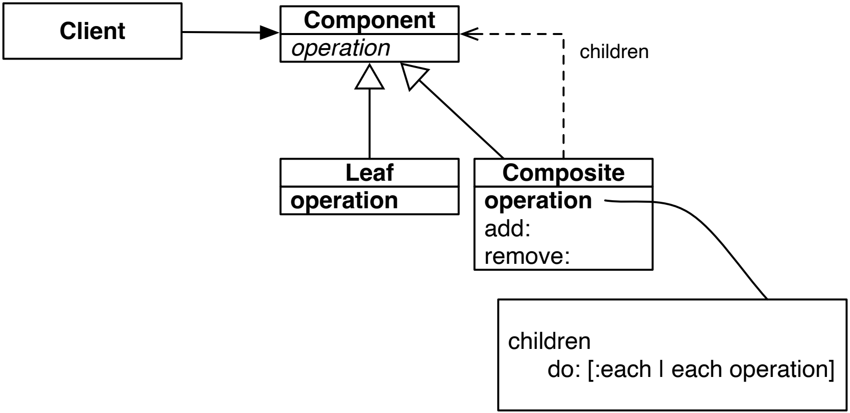
Elements and diagrams should offer the same API!



Composite: Intent

- Compose objects into tree structures to represent **part-whole** hierarchies
- Composite lets clients treat **individual** objects and **compositions** of objects **uniformly**

Composite design essence



Composite design essence

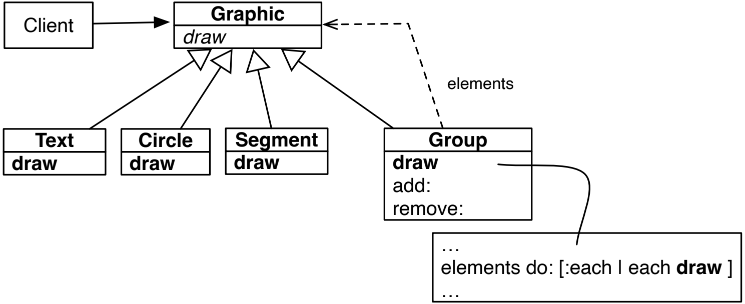
What is key:

- Leaves offers the **same** API than the composite
- Each leave will do something **different** but with the **same** API (polymorphism)
- Composite will offer the same API and some functionality to manage children

This brings **substituability** between the parts and the composite!

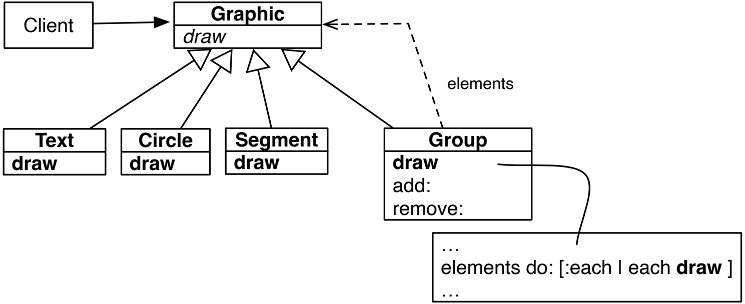
- Clients do not have to care

Composite participants: Client



Client manipulates objects in the composition through the **Component** interface (here Graphic)

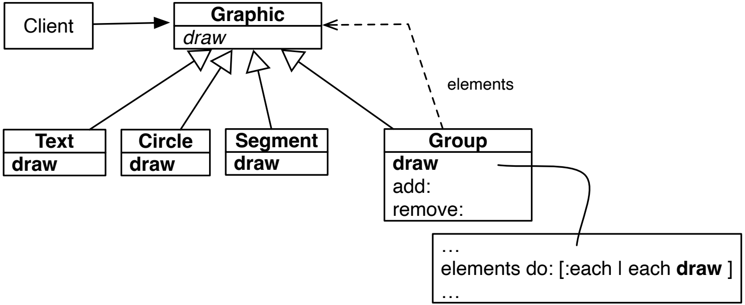
Composite participants: Component



Component (here Graphic)

- declares the interface for objects in the composition
- **may** implement default behavior for common interfaces
- **may** declare an interface for accessing and managing its child components

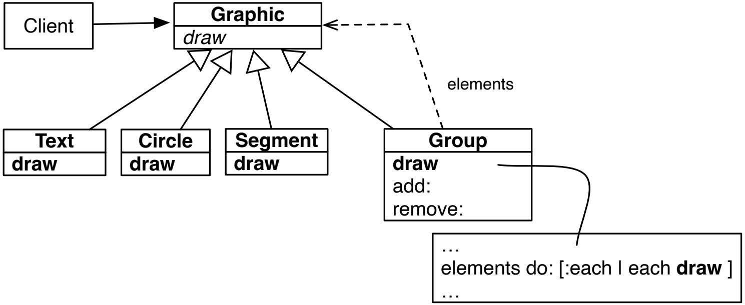
Composite participants: Leaf



Leaf (here Circle, Segment, Text, **...**)

- represents leaf objects in the composition.
- has usually no children
- defines behavior for primitive objects in the composition using a **polymorphic** API

Composite participants: Composite



Composite (here Group)

- defines behavior for components with children via a **polymorphic** API (here draw)
- stores child components
- implements child-related operations (add/remove...)

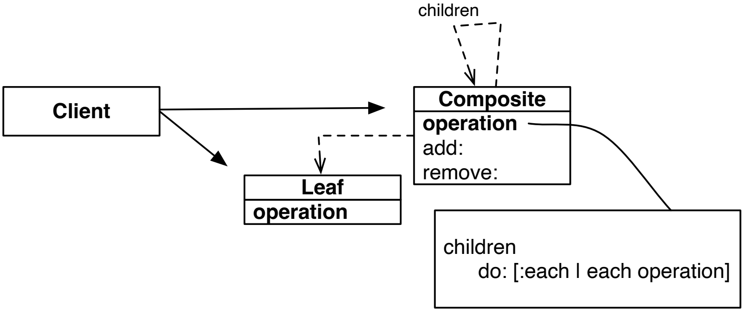
Composite consequences

- Defines class hierarchies consisting of primitive and composite objects exposing a common polymorphic API
- Clients do not have to explicitely check: Composite and leaves objects are treated uniformly
- Adding new leaves is simple

In dynamically-typed languages

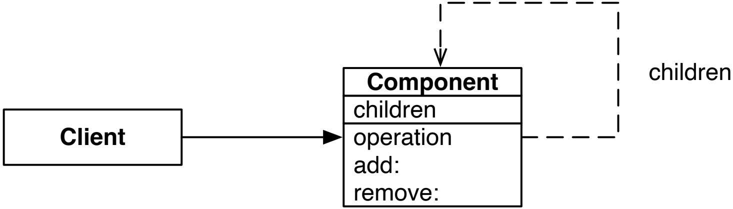
Polymorphism is expressed as classes exposing compatible API not compiled-time type check

- A composite and leaves do **not** have to inherit from a common ancestor
- Having a common ancestor eases understanding the composite, but it not mandatory



Alternate extreme implementation

- A Design Pattern is a name + intent
- Its implementation can have multiple forms



- Now the gain treating a leave as a container with a single element is unclear

Frequently Asked Questions

Can Composite contain any type of child?

- Yes
- Now the domain may impose some constraints
- And the implementation can enforce at the composite level

Can the Composite’ s number of children limited?

- Again it can be possible to control

Can we have different Composites within the same system?

- Yes and each Composite can have a different constraints, behavior, ..., delagating behavior

About Composite behavior

Forward/Delegation

- **Simple forward.** Send the message to all the children and merge the results without performing any other behavior
- **Selective forward.** Conditionally forward to some children
- **Extended forward.** Extra behavior
- **Override.** Instead of delegating

Composite and other design patterns

Composite and Visitors: Visitors walk on structured recursive objects e.g. composites
Composite and Factories: Factories can create composite elements

Conclusion

- Composite is a natural way of composing structural relationships
- Composite provides uniform API to clients
- Basis for complex treatment expressed as Visitor

A course by

S. Ducasse, G. Polito, and Pablo Tesone