Structural Patterns

From Gamma et al.

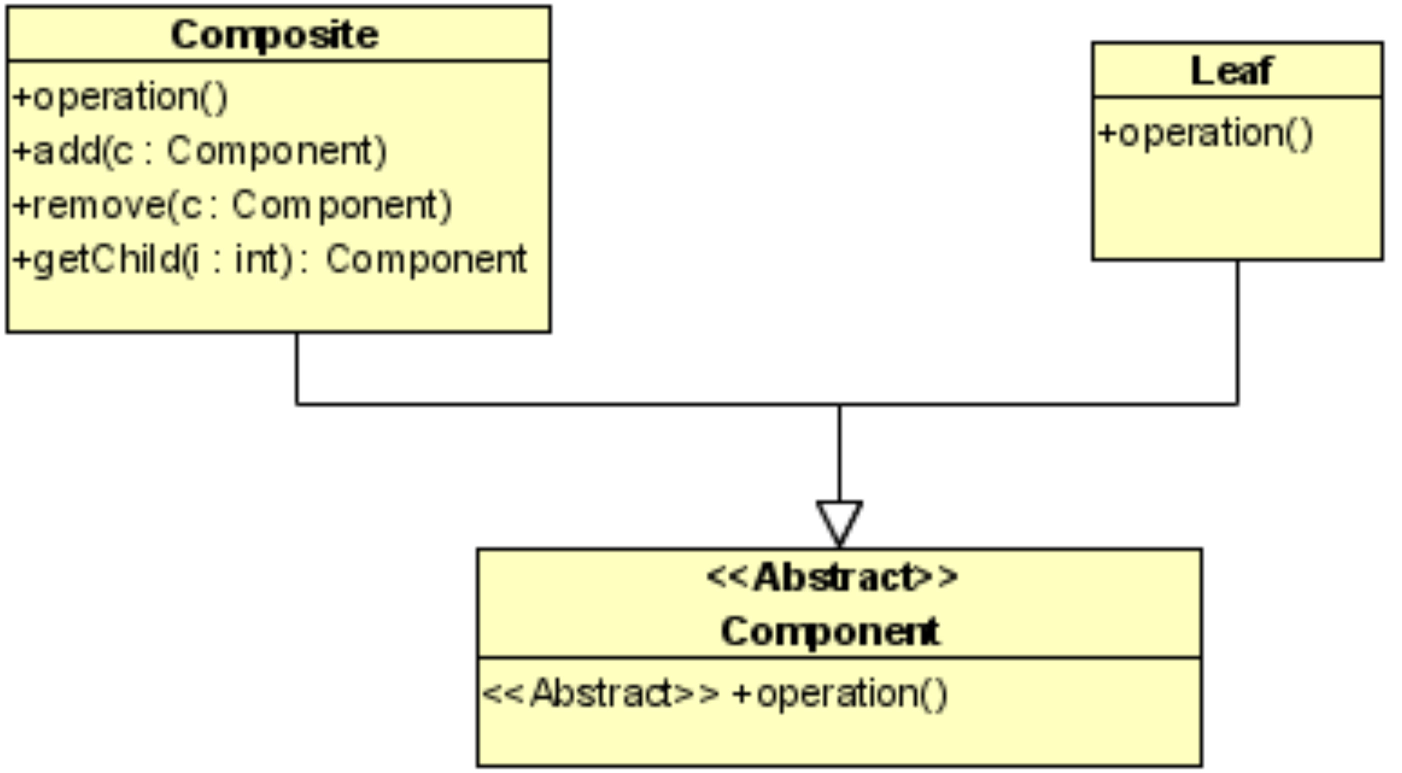
Behavioural, Structural, Creational Patterns

- Recall that patterns fit broadly into three categories: Behavioural, Structural, and Creational.
- The Observer, Command, and State patterns are Behavioural, they focus on behaviour
- Structural patterns focus on the relation between behaviour and structure.
- Next we look at two structural patterns: Composite and Proxy.

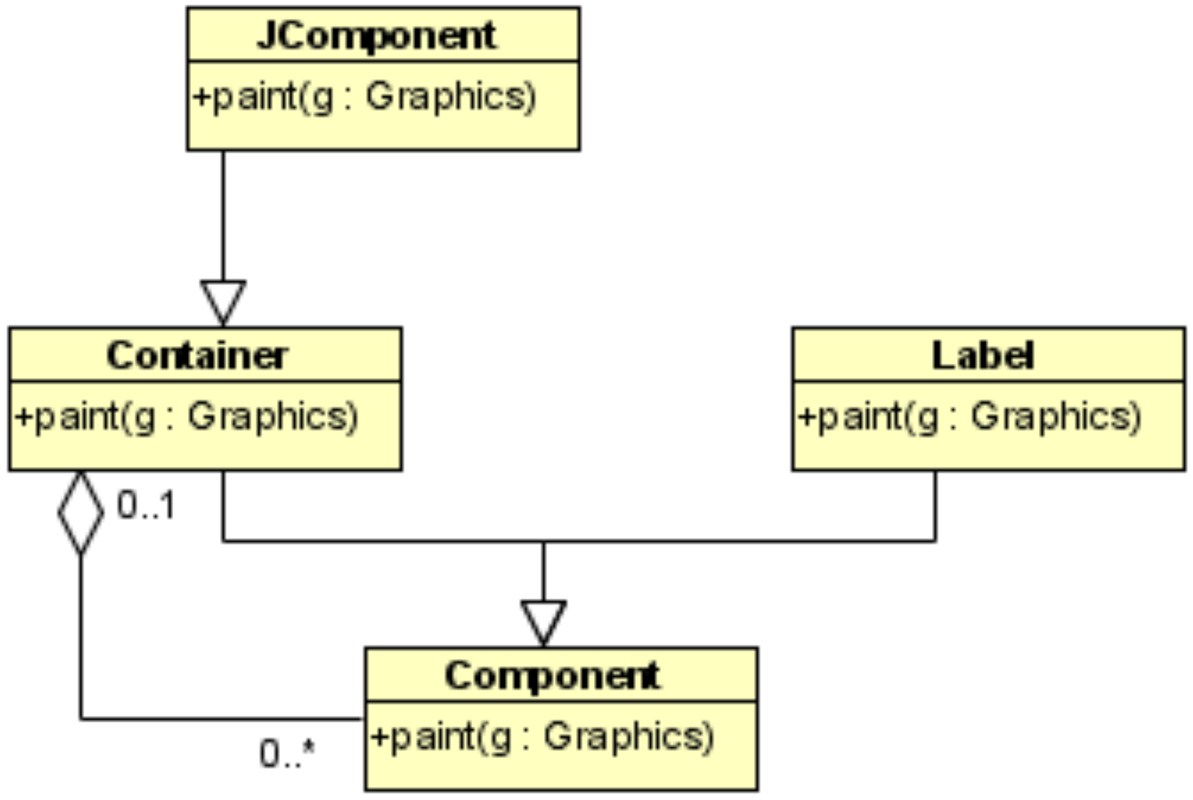
The Composite Pattern

- Idea: “Compose objects into trees structures to represent part-whole hierarchies”
- Example: AWT:
  - Visible widgets are represented by Components.
  - Components may be Containers.
  - So that Components form a tree.
  - Containers combine multiple Components
  - Each Container provides a coordinate system for its immediate descendants.
  - A call to “paint” (for example) on any container is forwarded recursively down to descendants.

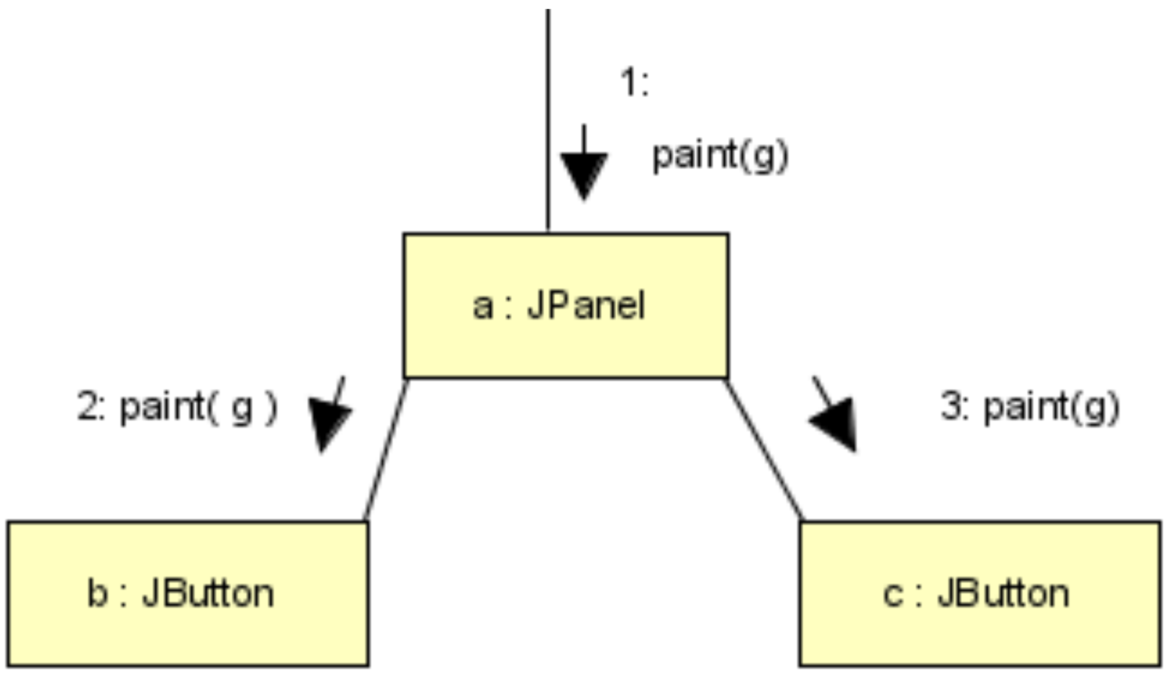
General relationships



Composite Pattern in AWT

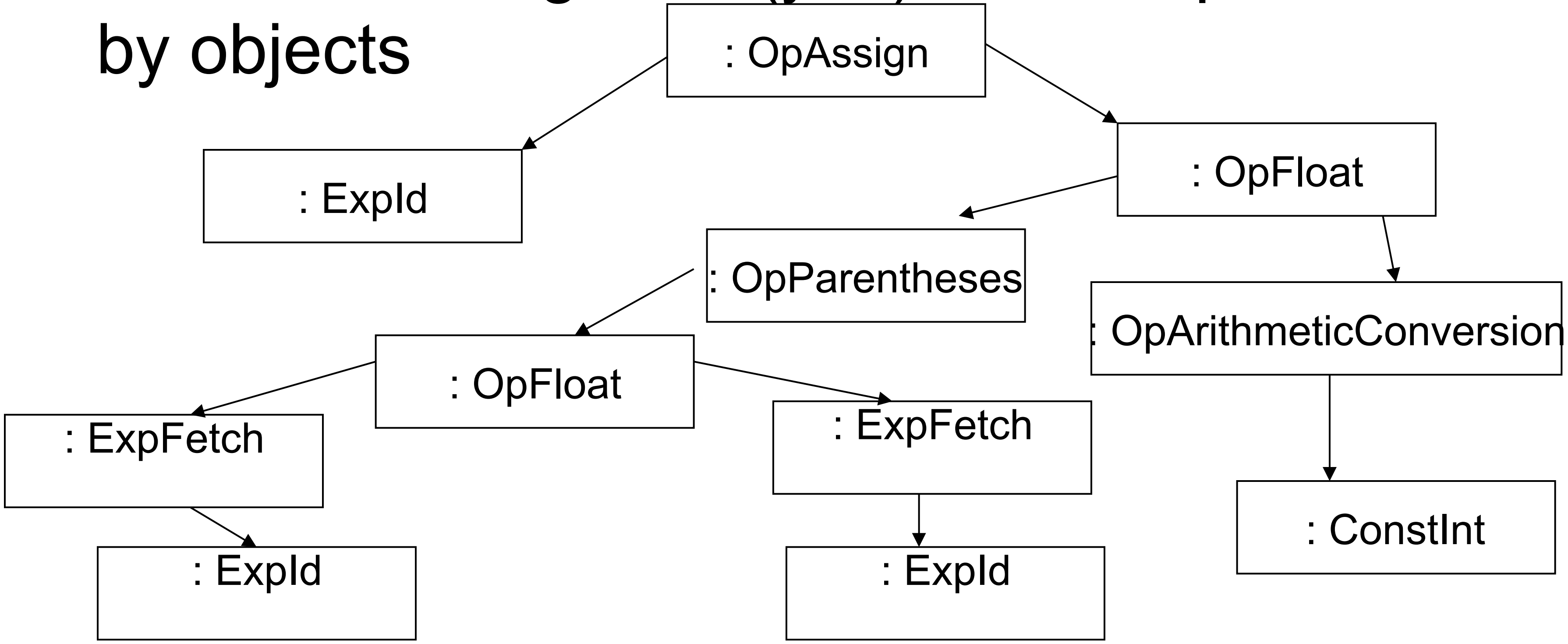


Delegation to child



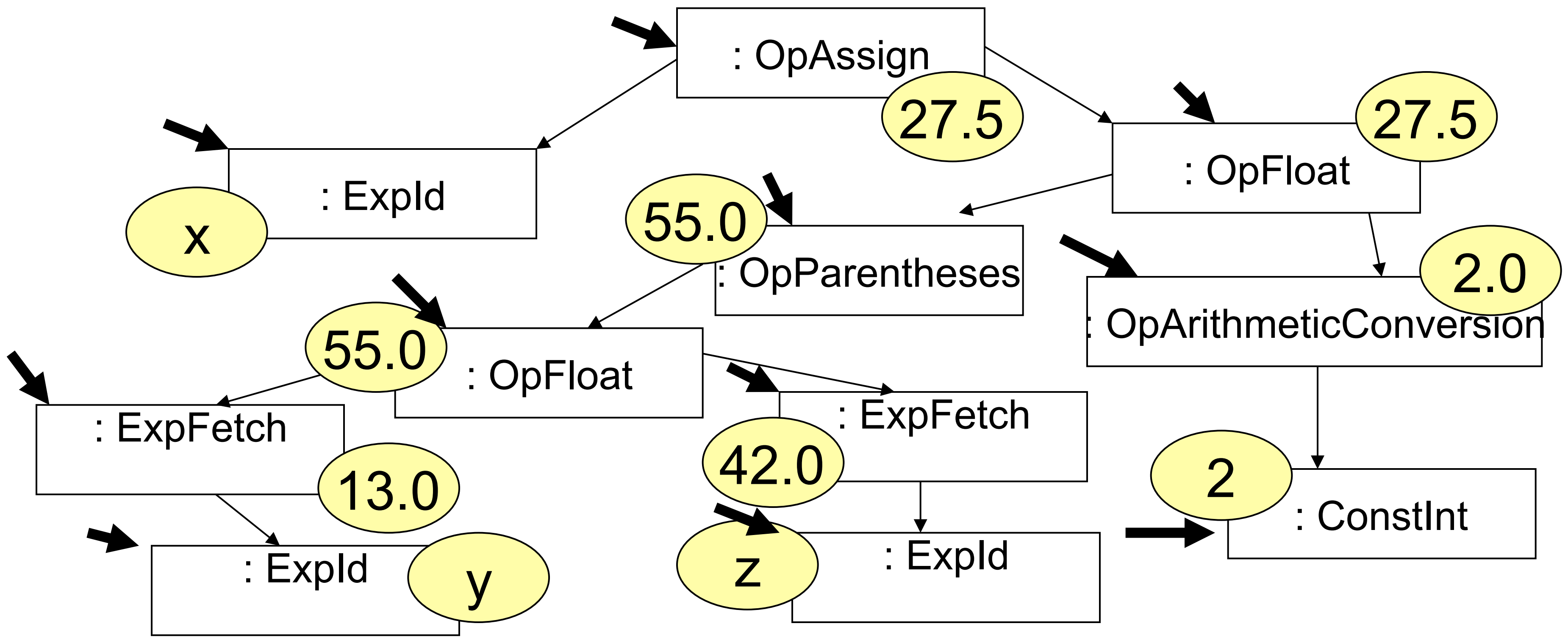
Example from the Teaching Machine

- Expressions are represented by nodes that form a tree. E.g. “x = (y+z) / 2” is represented by objects



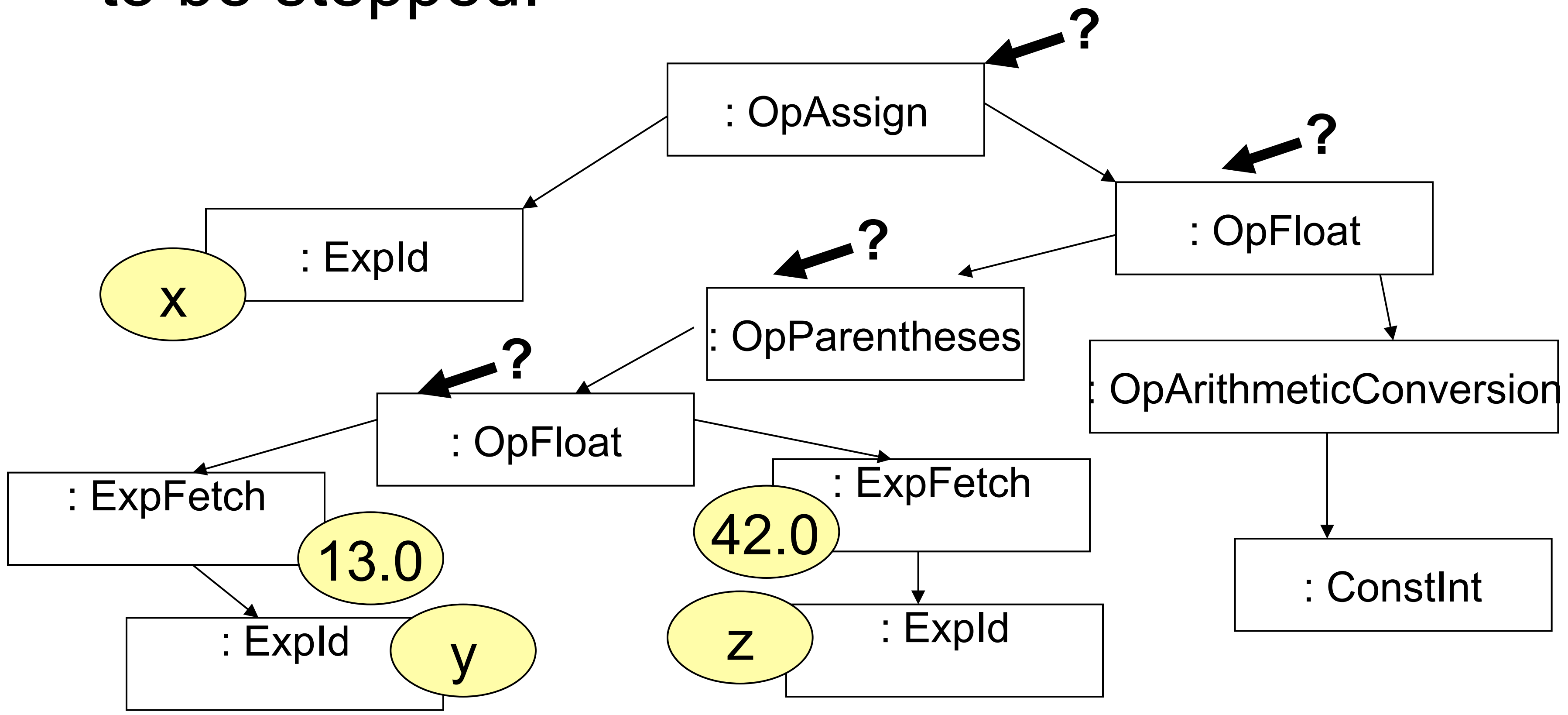
Example from the Teaching Machine

- Expressions are evaluated by alternately:
  - “Selecting” a ready node
  - “Stepping” the selected node



Selection in detail

- Selection starts with the root and is delegated downward until a node is found that is ready to be stepped.



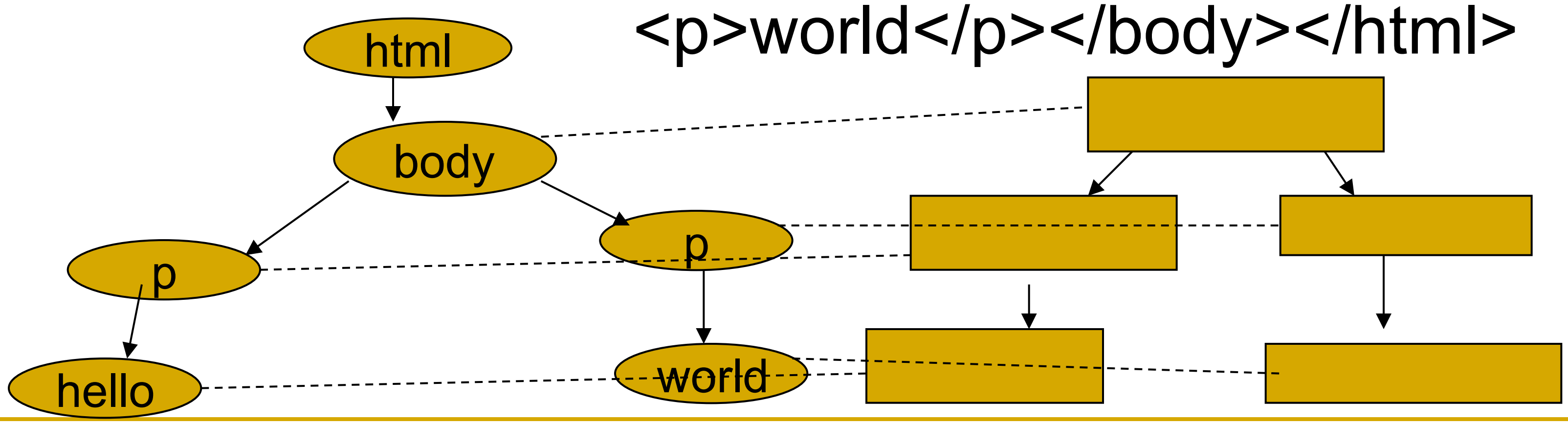
Selection strategies

- Left to right selection strategy
  - if all children of this node have been stepped
    - select this node
  - else
    - pick the left-most unstepped child
    - recursively delegate selection to that child
- Conjunction (&&) selection strategy
  - if the left child hasn't been stepped
    - recursively delegate selection to the left child
  - else if left child's is false or right child has been stepped
    - select this node
  - else
    - recursively delegate selection to the right child

Element trees in swing.text

Structured documents such as HTML pages are represented by Element objects.

- Each element is either a branch or a leaf
- Each element is associated with views
- E.g. <html><body><p>hello</p>
  <p>world</p></body></html>



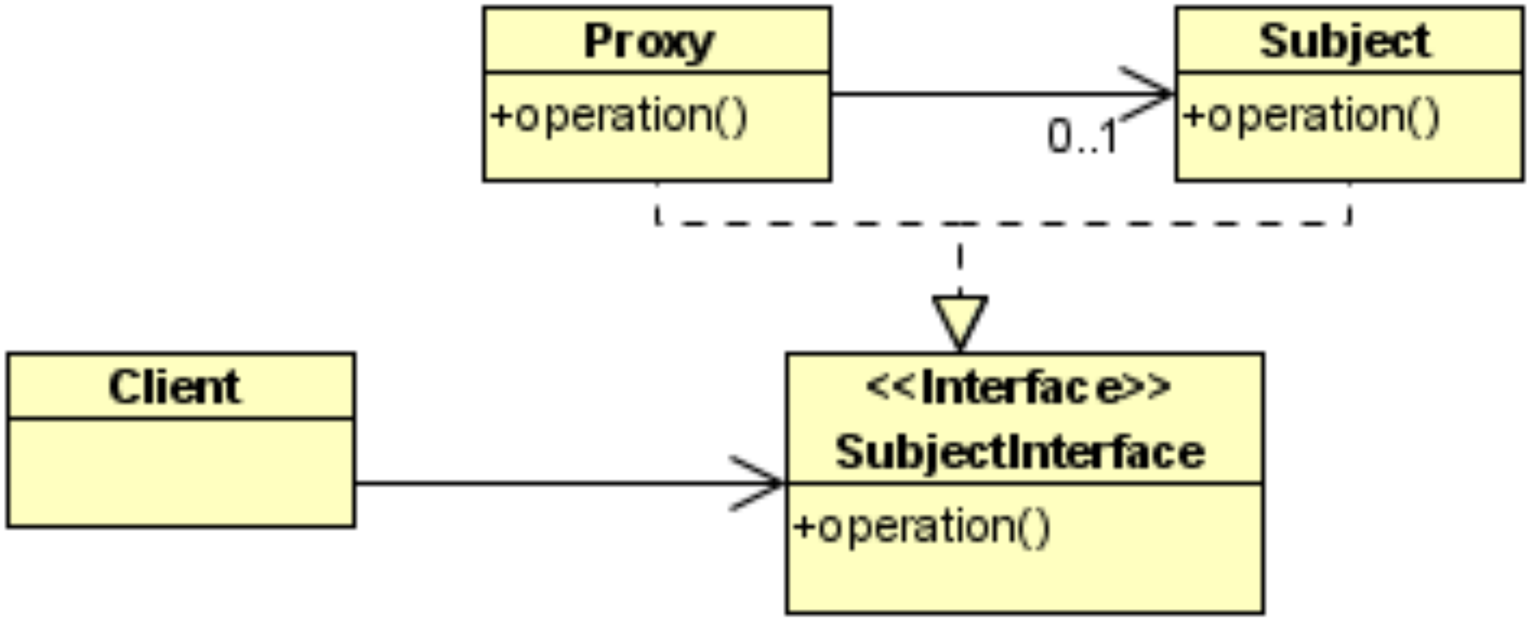
Composite Pattern: Consequences

- (+) Client need deal only with root: Existence of children can be hidden.
- (+) Easy to extend design by defining new kinds of nodes.

Proxy Pattern

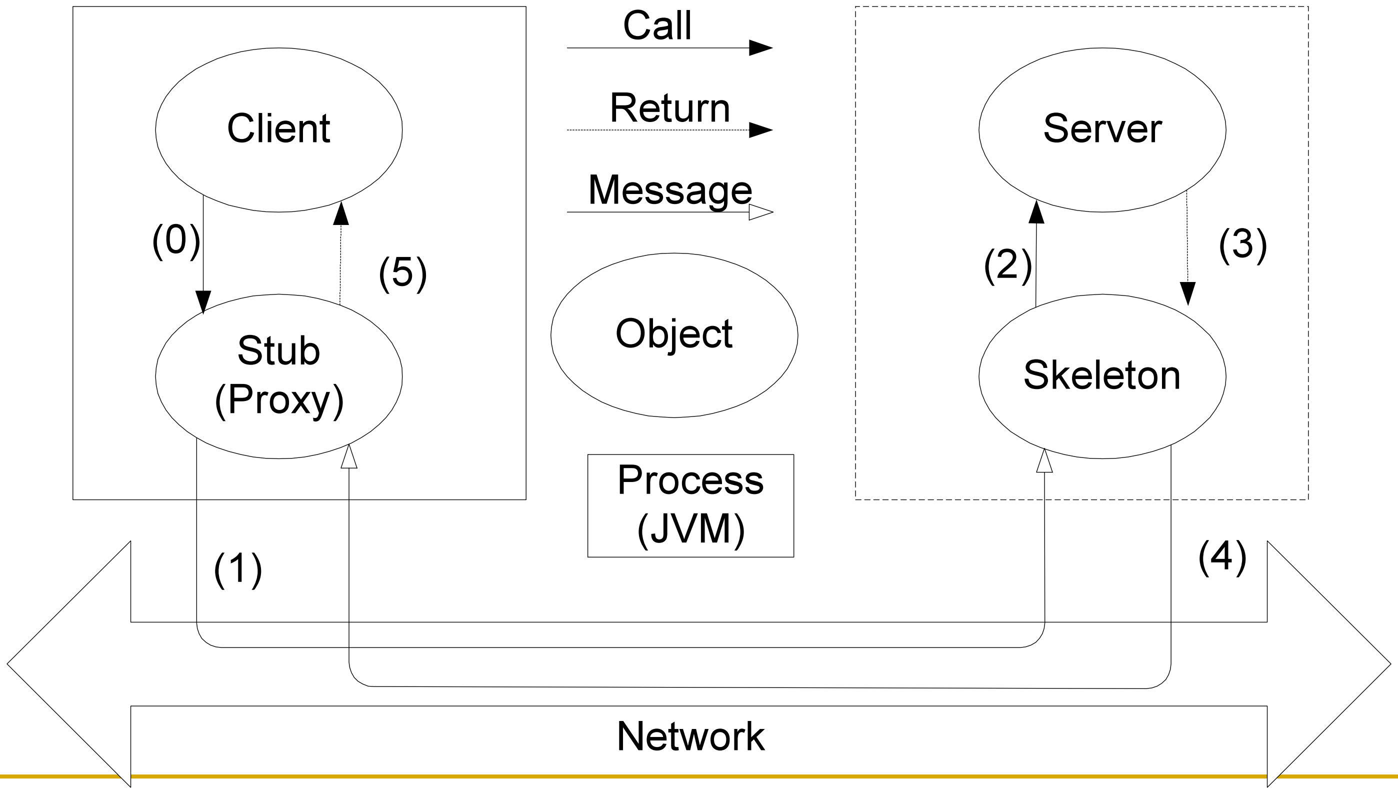
- Idea: Provides a surrogate or placeholder for another object.
- The Subject object and the Proxy object implement the same interface.
- The client typically does not depend on whether it is using the subject or the proxy.
- Proxy can be used to hide latency, hide remoteness, cache results, add functionality.

Proxy Pattern Structure



Example: Remote Method Invocation in Java (java.rmi)

- RMI allows client objects on one (virtual) machine to call methods of server objects on another machine.



Java RMI

- A program automates production of stubs and skeletons.
- A registry process helps clients find servers.
- Syntactic interface to proxy is identical to interface to server.
- Semantic interface is almost the same.

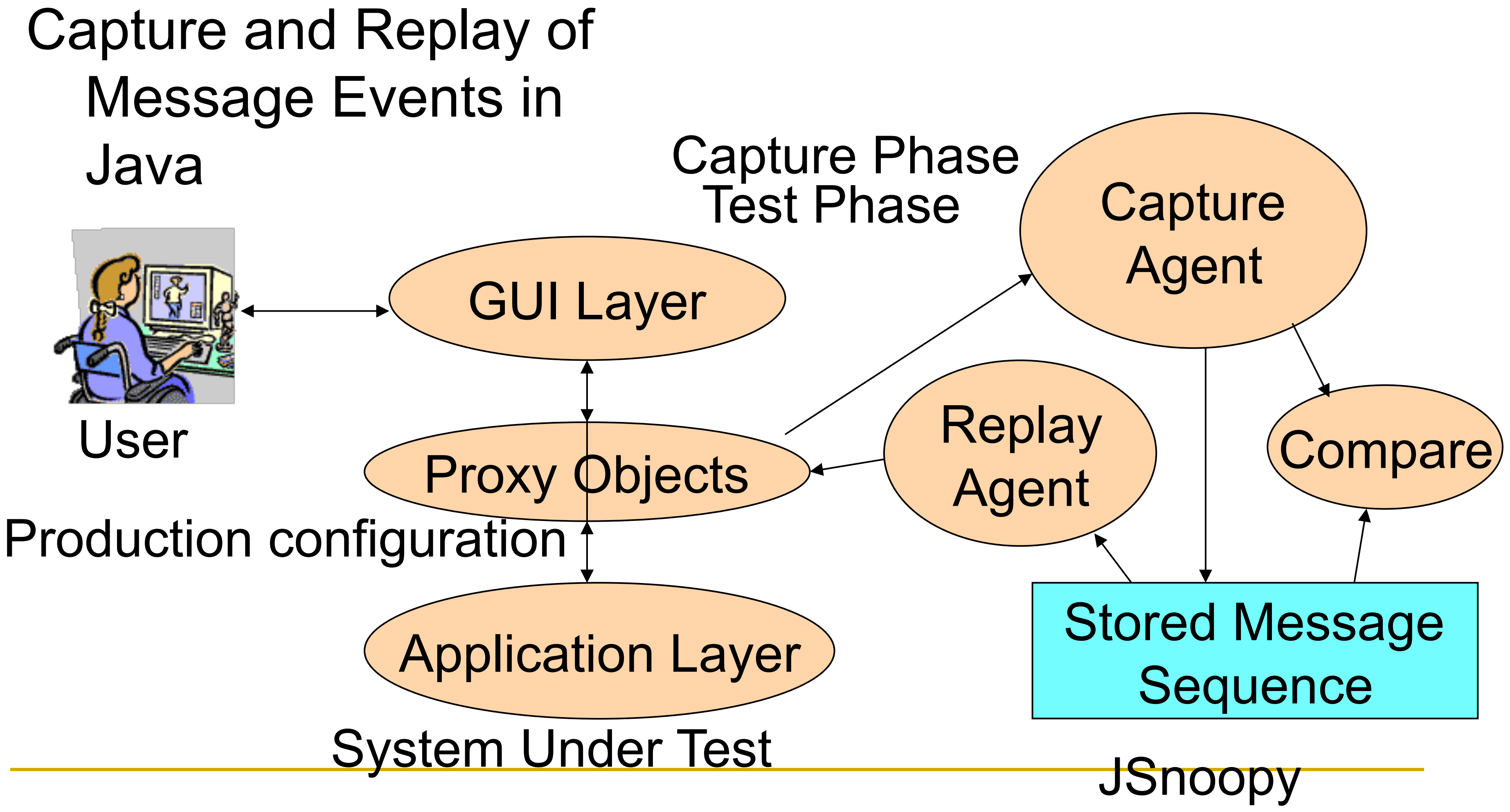
Example JSnoopy testing framework

- Goal: Allow automated regression testing of GUIs.
- Idea: Use proxy objects to observe all information between the GUI and the underlying application.
- Information means: messages (recipient, method, arguments), returned values, and exceptions.
- Capture:
  - A test is performed by a human tester via the GUI.
  - The tester visually verifies the software is working.
  - All information is captured and saved to file.

Example JSnoopy testing framework

- Now the software is changed and we need to check that what worked before still works.
- Replay:
  - Previously captured calls are injected via the proxies.
  - All information is captured.
  - The newly captured information is compared to the previously captured information.
  - Discrepancies are reported.
- The application layer and the GUI layer are indifferent to the existence of proxies.
- Java makes it easy to create these proxy objects (See java.lang.reflect.Proxy)

JSnoopy


Capture and Replay of Message Events in Java
Capture Phase
Test Phase
User
GUI Layer
Proxy Objects
Production configuration
Application Layer
System Under Test
Capture Agent
Replay Agent
Compare
Stored Message Sequence
JSnoopy

JSnoopy

- Arguably the JSnoopy example is an example of the Decorator pattern
- In the Decorator pattern new responsibilities are added to the underlying subject.
- In the Proxy pattern the service with and without the proxy is the same.
- The line is fuzzy.

Proxy Pattern: Consequences

- (+) Proxies add a layer of functionality without affecting either the client or the subject.
- (+) Proxy can hide the differences between communicating with local vs. remote objects.
- (+) Proxies can hide latency. E.g. by providing minimal functionality before the real subject is loaded.
- (-) Use of proxy may change semantics of calls.