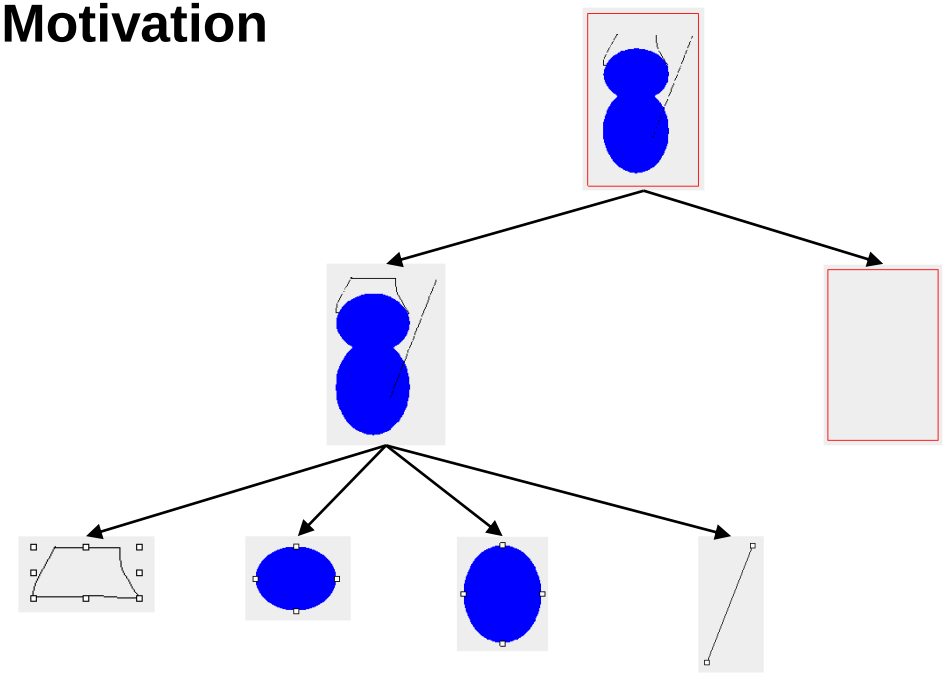
Composite Pattern

- **Intent**
  - Representation of recursive part-whole hierarchies
  - Individual objects and compositions of objects should be treated uniformly
- **Motivation**

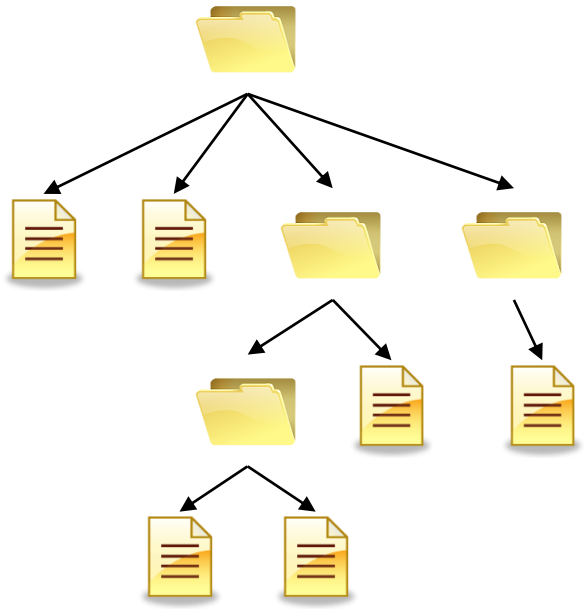


Operations applied uniformly on composites and single objects:

- move
- draw
- resize
- copy

Composite Pattern

- **Example: File Structures**

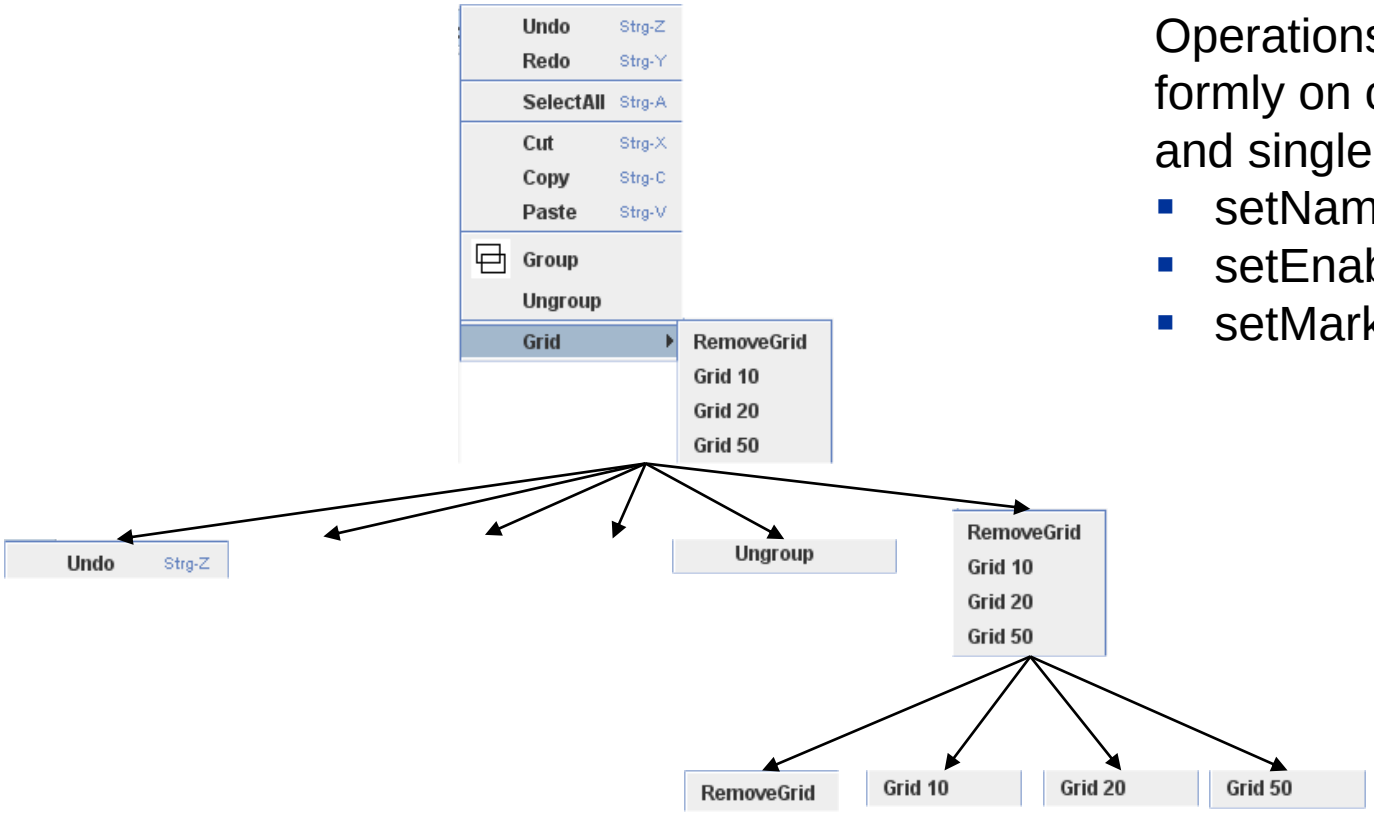


Operations applied uniformly on composites and single objects:

- getSize
- setProperties
- delete

Composite Pattern

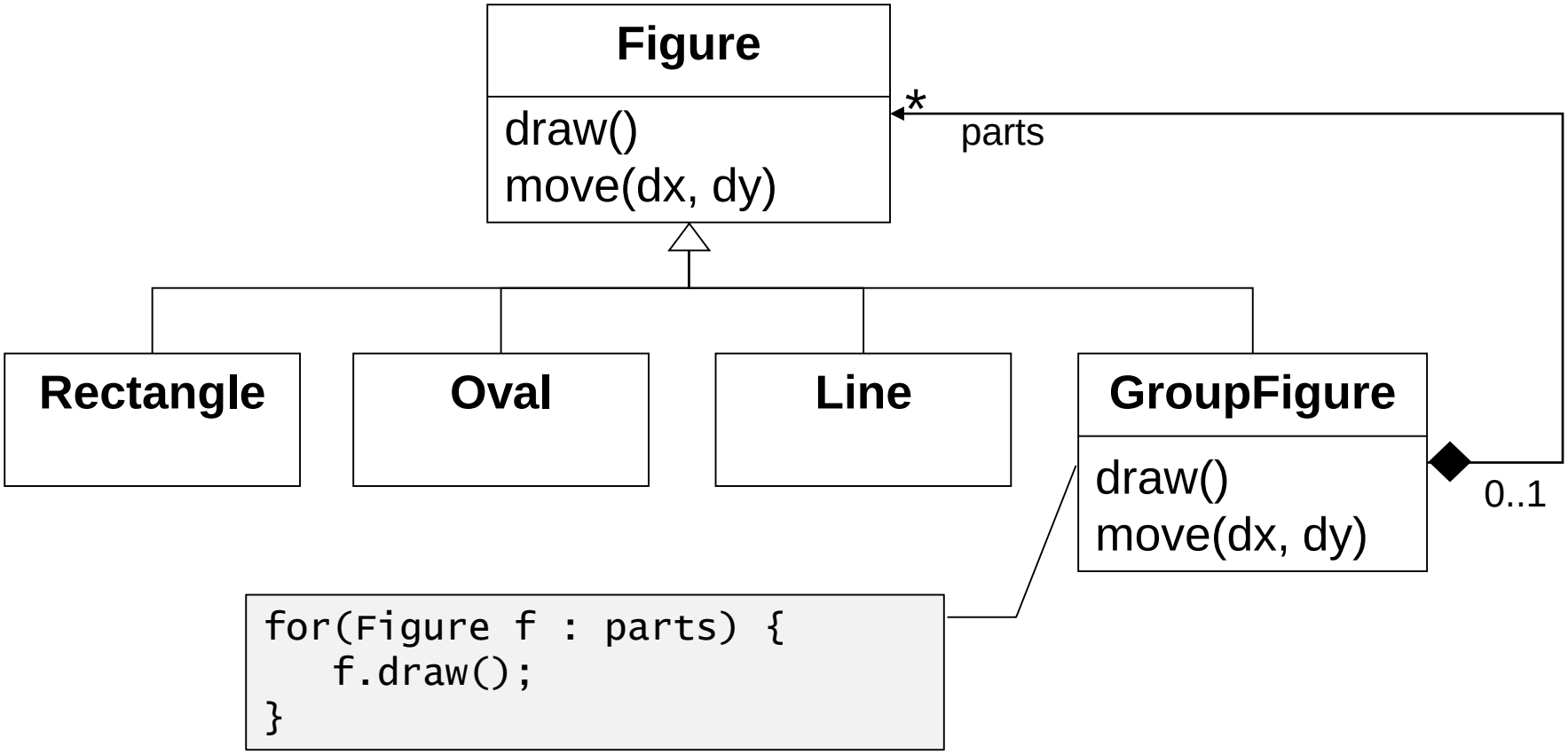
- **Example: Menus**



Operations applied uniformly on composites and single objects:

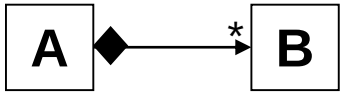
- setName
- setEnabled
- setMarked

Composite Pattern: Structure

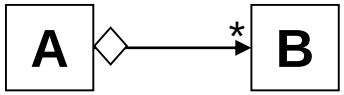


Composite Pattern: Structure

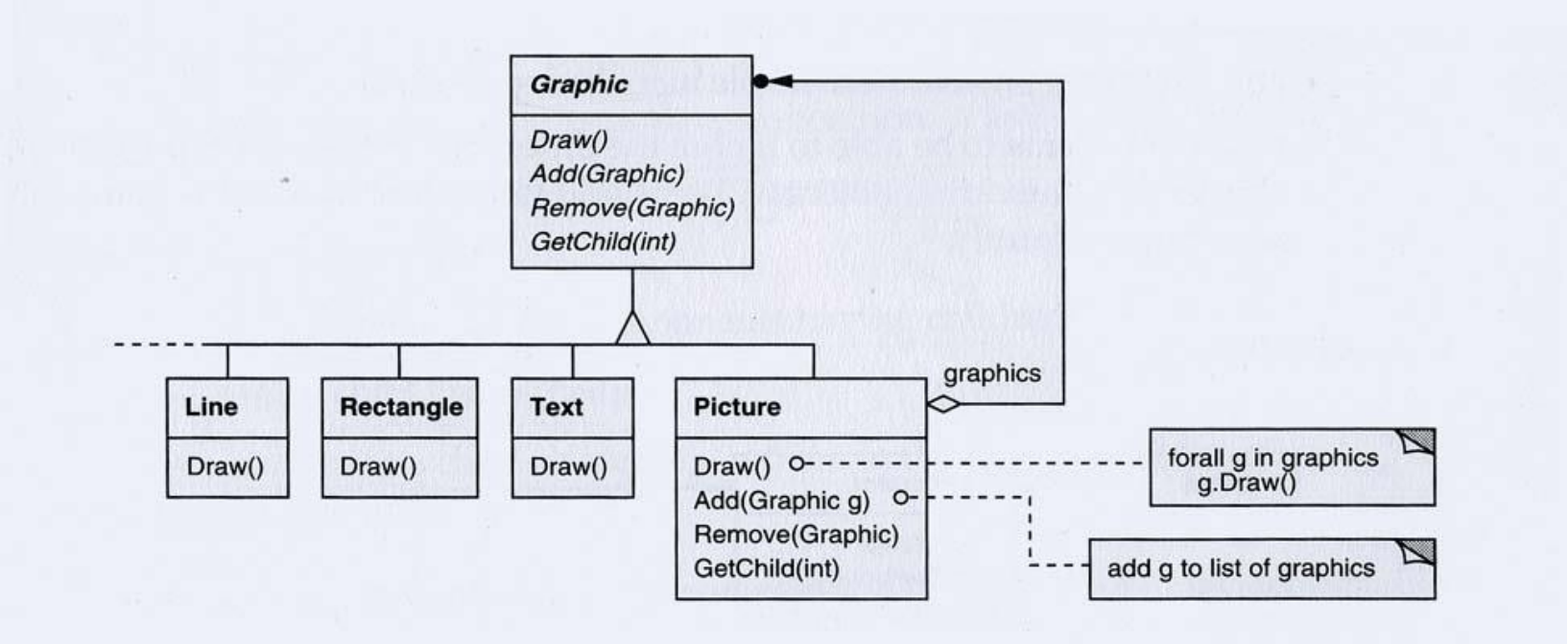
- **Composition**
  - B is a fixed part of A
    - Part may be part of at most one whole
    - If whole is deleted, then also all its parts
    - Whole acts substitutional for its parts, i.e. operations are propagated to its parts
  - Sample: Car
  - Tree structure



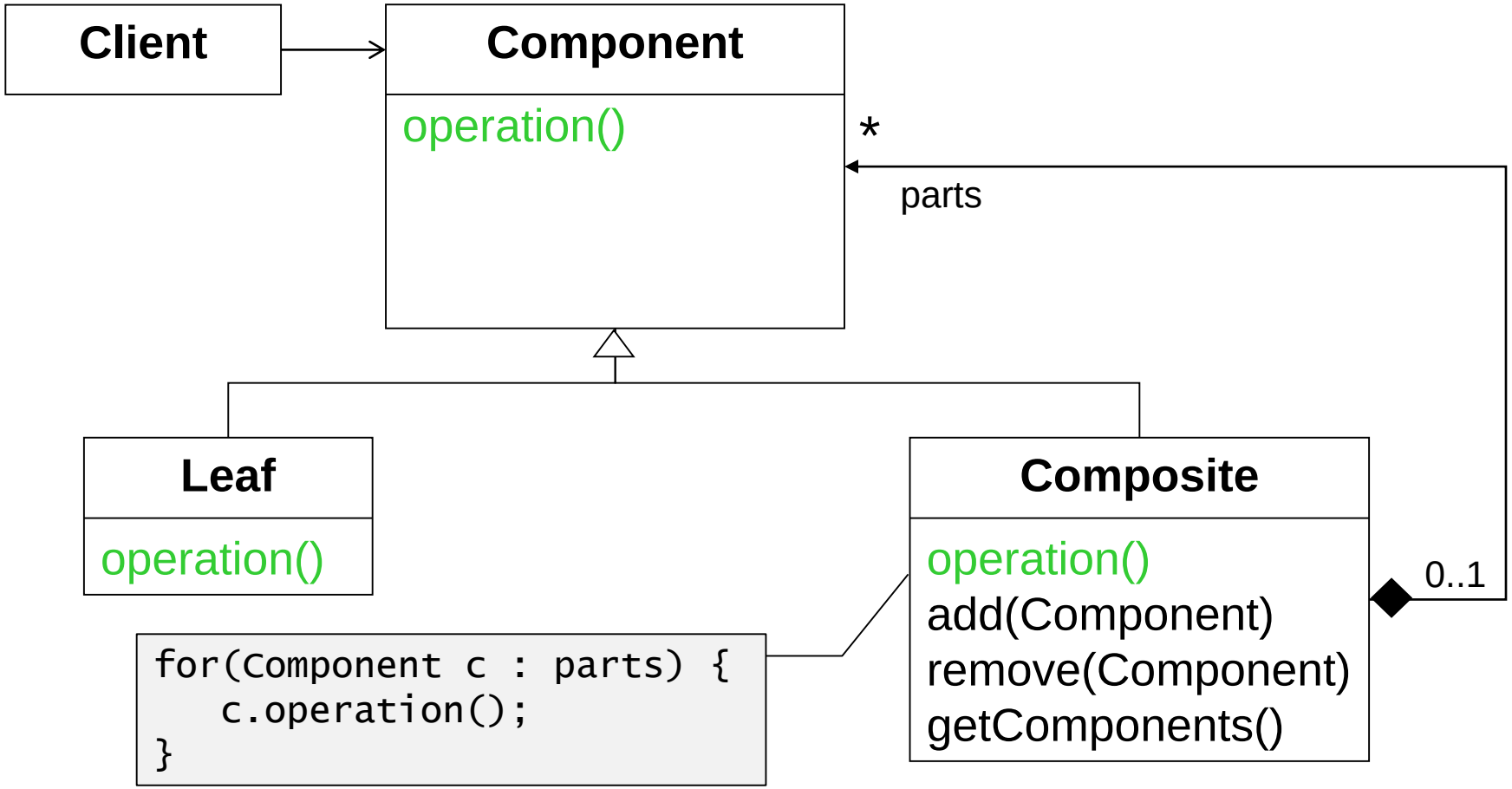
- **Aggregation**
  - B is a variable part of A
    - Part may be part of several wholes
    - Part may exist individually
  - Sample: Organizational structure
  - DAG structure



Composite Pattern: Structure



Composite Pattern: Structure



Composite Pattern: Participants

- **Component**
  - Declares the interface for the objects in the composition
  - Implements the default behavior
  - Optionally: declares methods for accessing and managing child objects
- **Leaf**
  - Defines behavior for primitive objects
- **Composite**
  - Stores children references
  - Defines behavior for components having children
  - Implements methods for accessing and managing child objects
- **Client**
  - Manipulates objects through the Component interface

Composite: GroupFigure

- **GroupFigures**

```
public class GroupFigure implements Figure {

   public GroupFigure(Figure... figures) {
      for (Figure f : figures) { addFigure(f); }
   }

   public void addFigure(Figure f) { ... }

   ....
}
```

Composite: GroupFigure

- **Problem 1:**

```
Figure f1 = new RectangleFigure( ... );
Figure f2 = new RectangleFigure( ... );
Figure f3 = new OvalFigure( ... );

GroupFigure g1 = new GroupFigure(f1, f2);

GroupFigure g2 = new GroupFigure(f3, f1);
```

Composite: GroupFigure

- **Problem 2:**

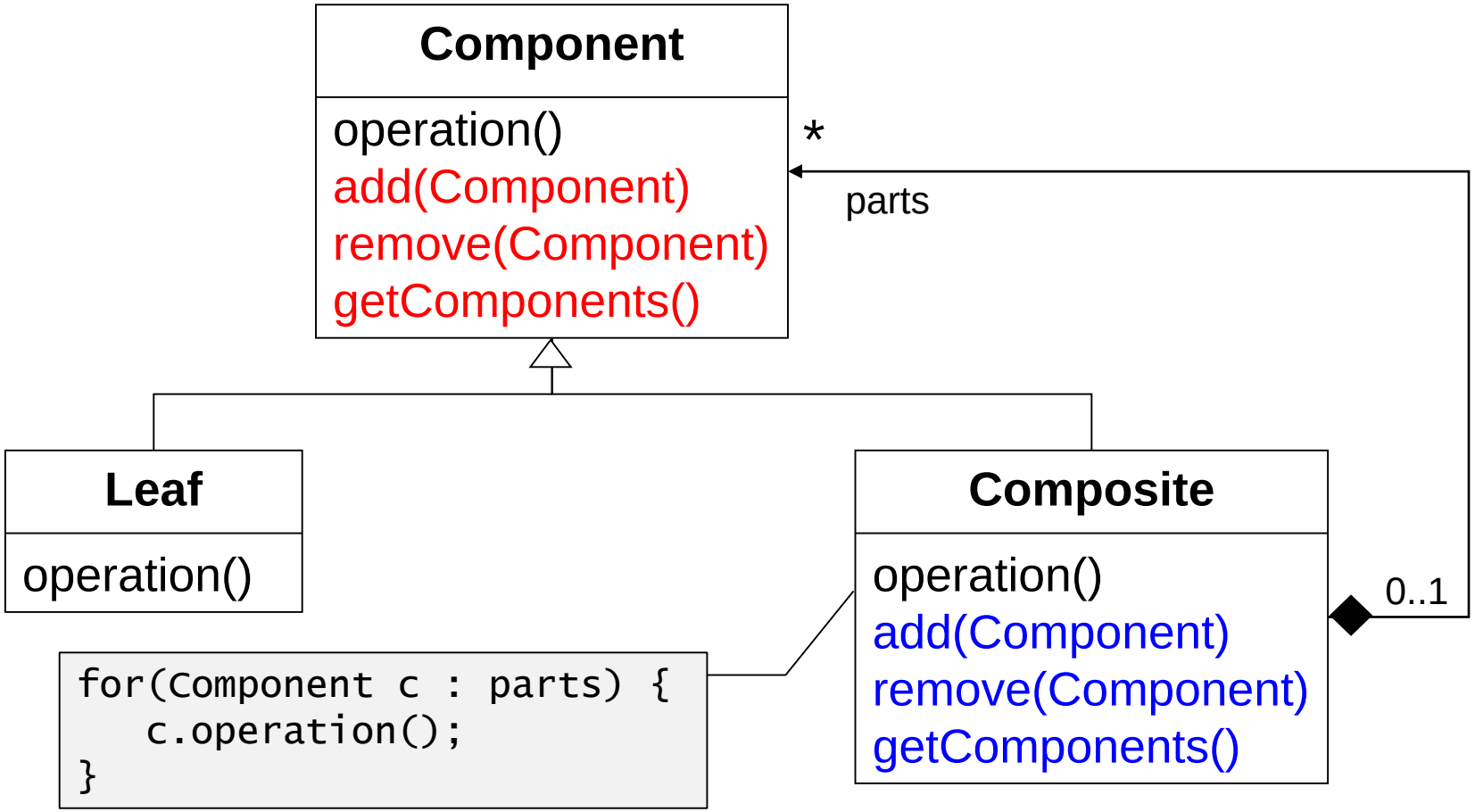
```
Figure f1 = new RectangleFigure( ... );
Figure f2 = new RectangleFigure( ... );
Figure f3 = new OvalFigure( ... );

GroupFigure g1 = new GroupFigure(f1, f2);

GroupFigure g2 = new GroupFigure(g1, f3);

g1.addFigure(g2);
```

Composite Pattern: Structure



Composite Pattern

- **Where should we define the Composite-specific methods ?**

```
public void add(Component c) { … }
public void remove(Component c) { … }
public List<Component> getComponents() { … } // immutable list
```

  - In Component

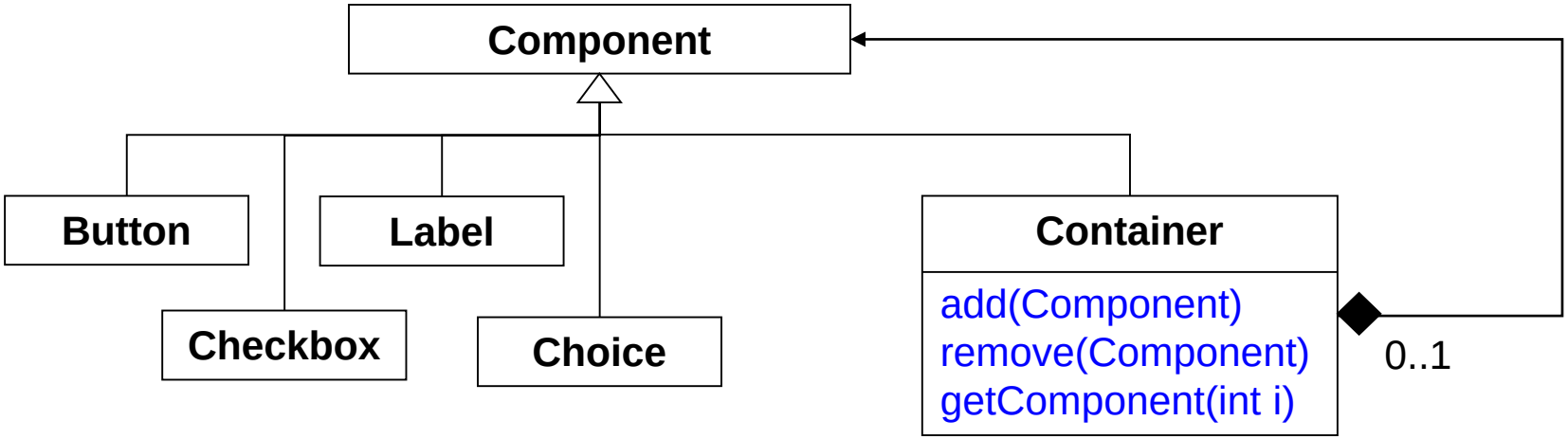
  - In Composite

TRANSPARENT

SAFE

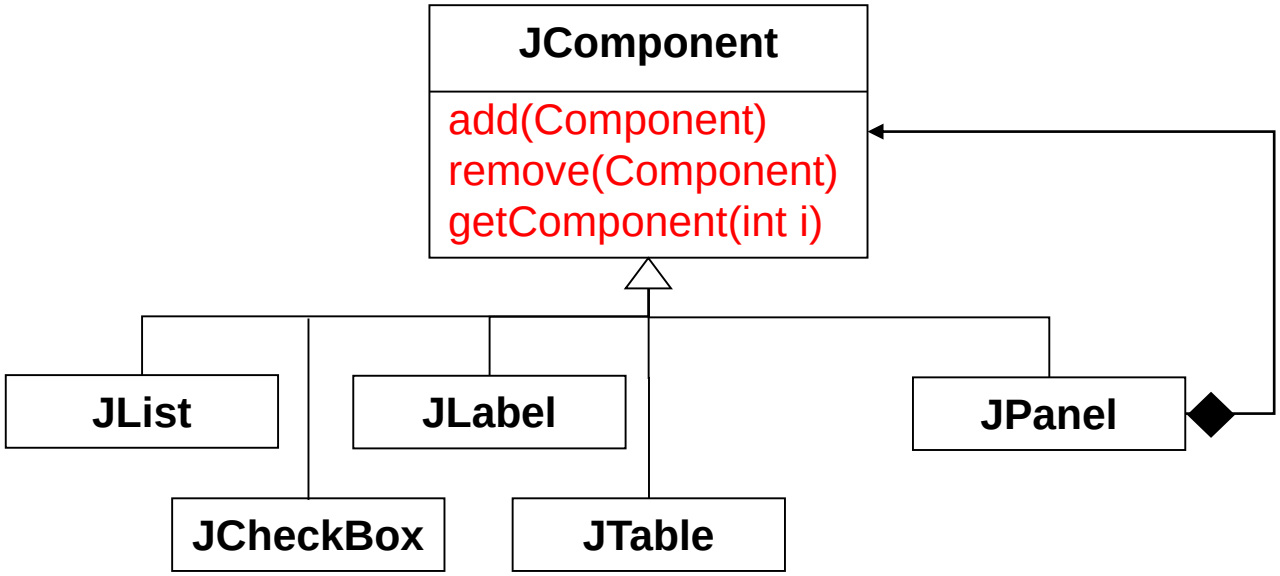
Composite Pattern: AWT

- **Example AWT**

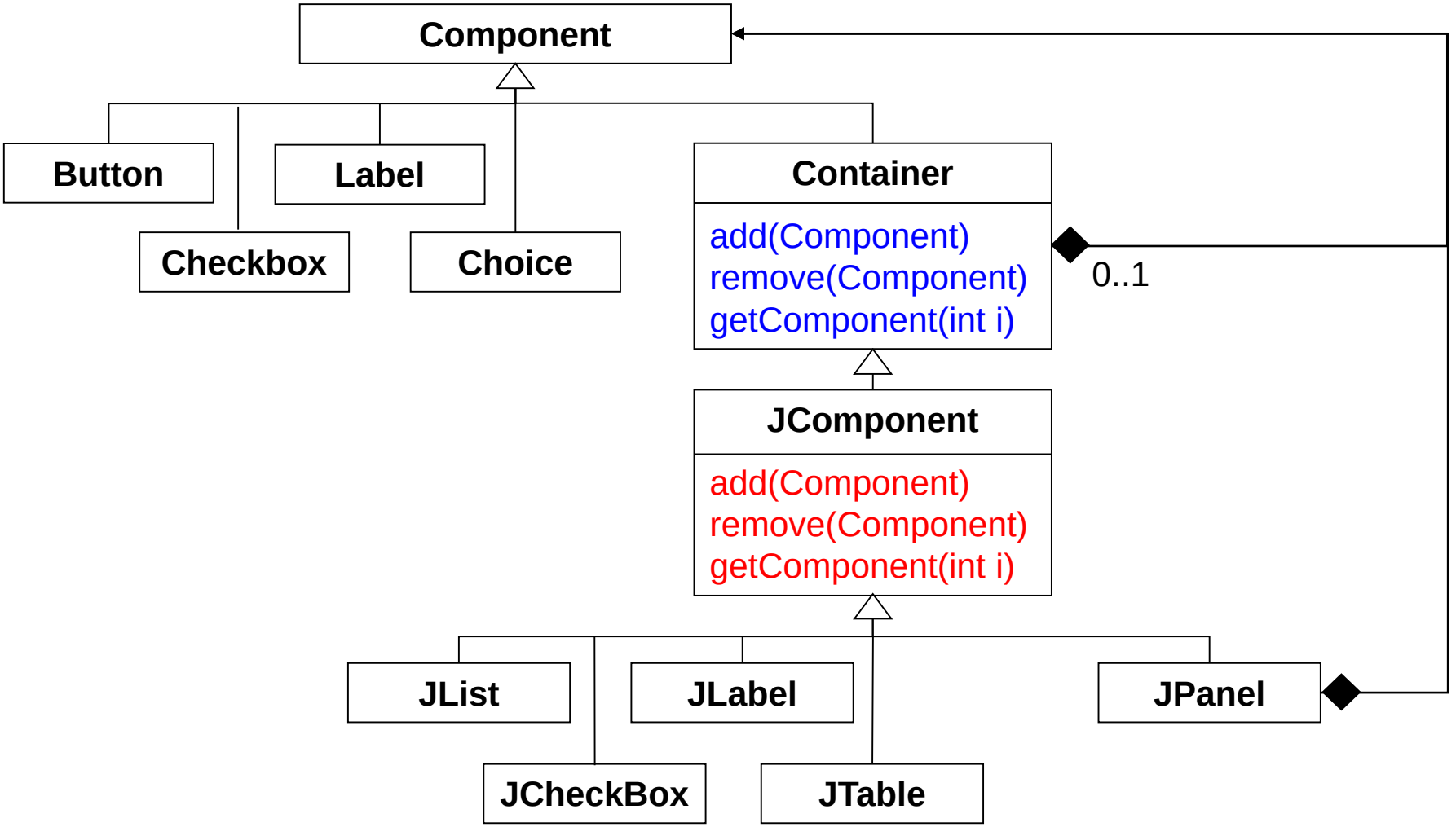


Composite Pattern: Swing

- **Example Swing**



Composite Pattern: AWT & Swing



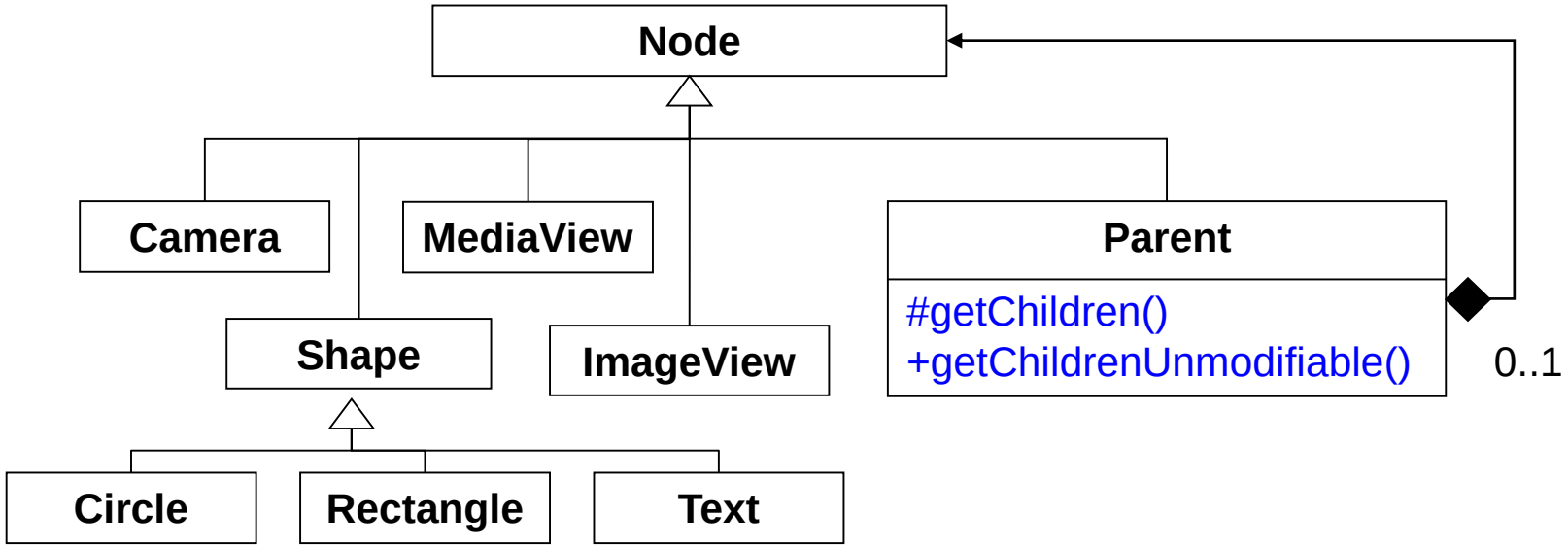
Composite Pattern: AWT & Swing

```
public class ButtonOnButton {
   public static void main(String[] args) {
      JButton b = new JButton("Button");
      b.setLayout(new FlowLayout());
      b.add(new JLabel("Label"));
      b.add(new JButton("Hallo"));

      JFrame p = new JFrame();
      p.add(b); p.pack(); p.setVisible(true);
   }
}
```

Composite Pattern: JavaFX Scene Graphs

- **javafx.scene.Node**
  - base class for scene graph nodes



Composite Pattern: JavaFX Scene Graphs

- **javafx.scene.Parent**
  - base class for all nodes that have children in the scene graph
  - getChildren is typically overridden to promote it to be public