Composite Pattern

Vladimir Martinka

IBM

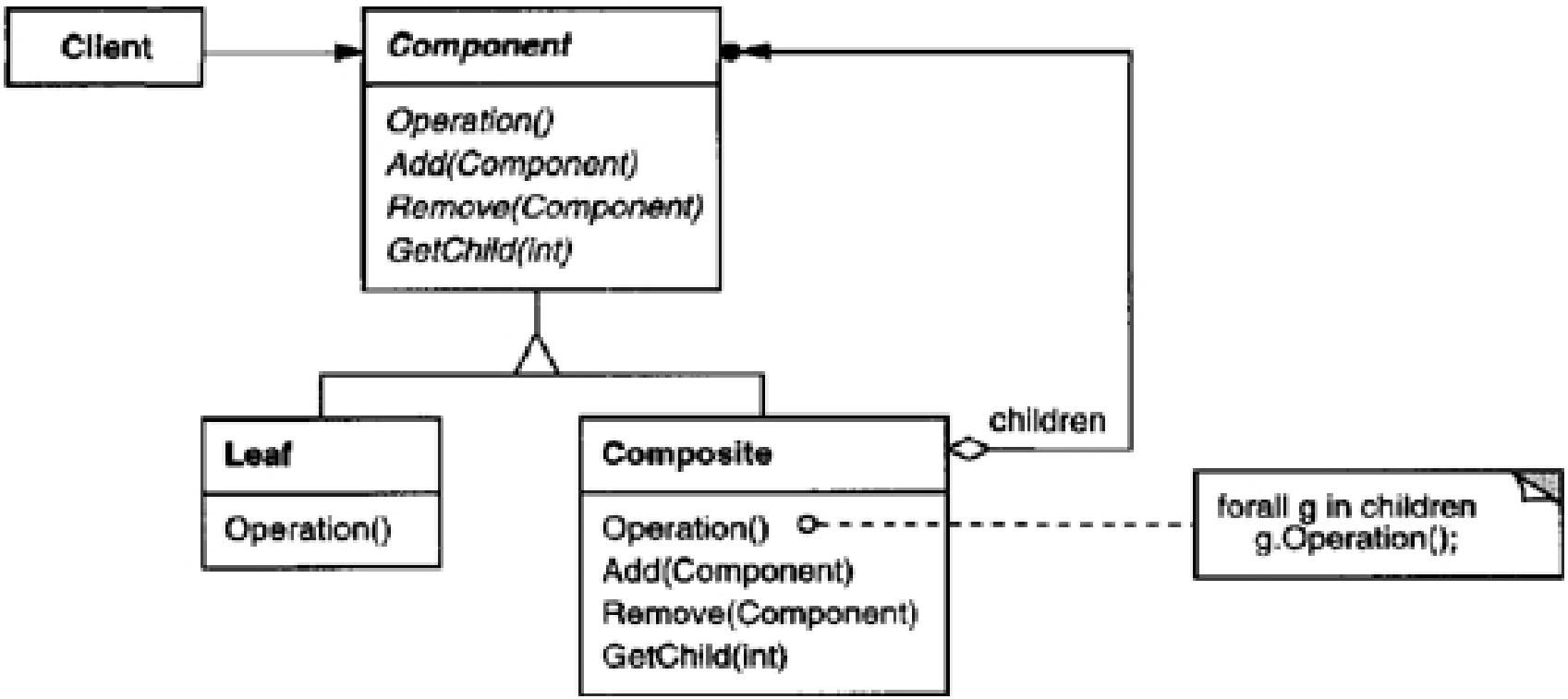
March 17, 2019

Definition

- Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly. [1]

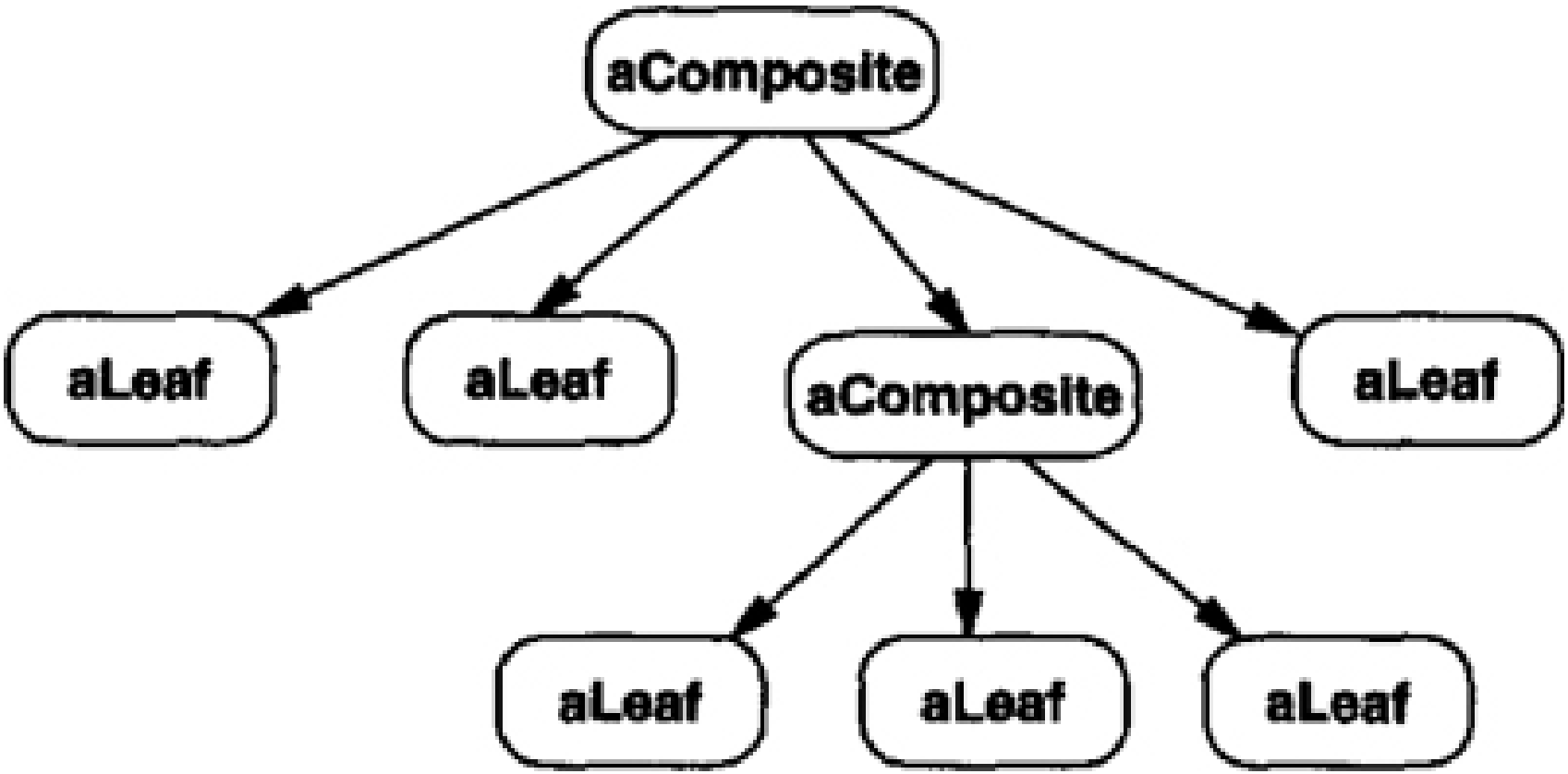
UML

Figure: Abstract structure of Composite pattern [1]



Example structure

Figure: Example composite object structure [1]



Usage

- Graphics (parent draws subcomponents)
- Tree-structured reports
- Shopping cart
- ...

Code example

Component interface

```
public interface CartItem {
    BigDecimal getPrice(); //operation

    void add(CartItem item);
    void remove(CartItem item);
    default List<CartItem> getChildren() {
        return Collections.emptyList();
    }
}
```

Code example

Composite

```
public class CartItemComposite implements CartItem {

    private final List<CartItem> children = new ArrayList<>();

    private final String name;

    public CartItemComposite(String name) {
        this.name = name;
    }

    @Override
    public BigDecimal getPrice() {
        return getChildren().stream().map(CartItem::getPrice).reduce(BigDecimal::add).get();
    }

    @Override
    public void add(CartItem item) {...3 lines }

    @Override
    public void remove(CartItem item) {...3 lines }

    @Override
    public List<CartItem> getChildren() {...3 lines }

    @Override
    public String toString() {...4 lines }
}
```

Code example

Leaf

```
public class Donut implements CartItem {

    private final String name;

    public Donut(String name) {
        this.name = name;
    }

    @Override
    public BigDecimal getPrice() {
        return BigDecimal.valueOf(80);
    }

    @Override
    public void add(CartItem item) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void remove(CartItem item) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public String getName() {
        return name;
```

Code example

Client

```
    private static void testGofStyleComposite() {
        CartItem root = new CartItemComposite("total");

        CartItem donutCombo = new CartItemComposite("donutCombo");
        donutCombo.add(new Donut("Strawberry"));
        donutCombo.add(new Donut("Chocolate"));

        CartItem breakfast = new CartItemComposite("breakfast");
        breakfast.add(donutCombo);
        breakfast.add(new Cola());

        root.add(breakfast);

        root.add(new Donut("Snack"));

        System.out.println(root);
        System.out.println(breakfast);

        try {
            donutCombo.getChildren().get(0).add(new Cola()); //will fail - cannot add items to donut
        } catch (UnsupportedOperationException e) {
            e.printStackTrace(System.out);
        }
    }
```

```
mposite.Client > testGofStyleComposite >
t - Composite (run) | Notifications | Action Items | Bookmarks | Usages
run:
340 $ for total { 260 $ for breakfast { 160 $ for donutCombo { Donut Strawberry : 80 $, Donut Chocolate : 80 $ }, Cola : 100 $ }, Donut Snack : 80 $ }
260 $ for breakfast { 160 $ for donutCombo { Donut Strawberry : 80 $, Donut Chocolate : 80 $ }, Cola : 100 $ }
java.lang.UnsupportedOperationException: Not supported yet.
        at composite.gofapproach.Donut.add(Donut.java:29)
        at composite.Client.testGofStyleComposite(Client.java:47)
        at composite.Client.main(Client.java:23)
```

Summary

[1] 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.

Watch out for

- Violation of single responsibility principle
- Design can get too general, client has no control over implementation, if that becomes necessary, run time checks may be required
- Children do not support all common operations - requires handling with some trade-offs in either safety or transparency

References I


R. H. E. G. John Vlissides, Ralph Johnson.
Design Patterns: Elements of Reusable Object-Oriented Software, chapter 4.
Addison-Wesley Professional, 10 1994.


V. Martinka.
Latex for this presentation.
`https://www.overleaf.com/read/mcpnfddcmhbk`, 3 2019.


V. Martinka.
Source codes for this presentation.
`https://github.com/vladimir-martinka/Composite`, 3 2019.