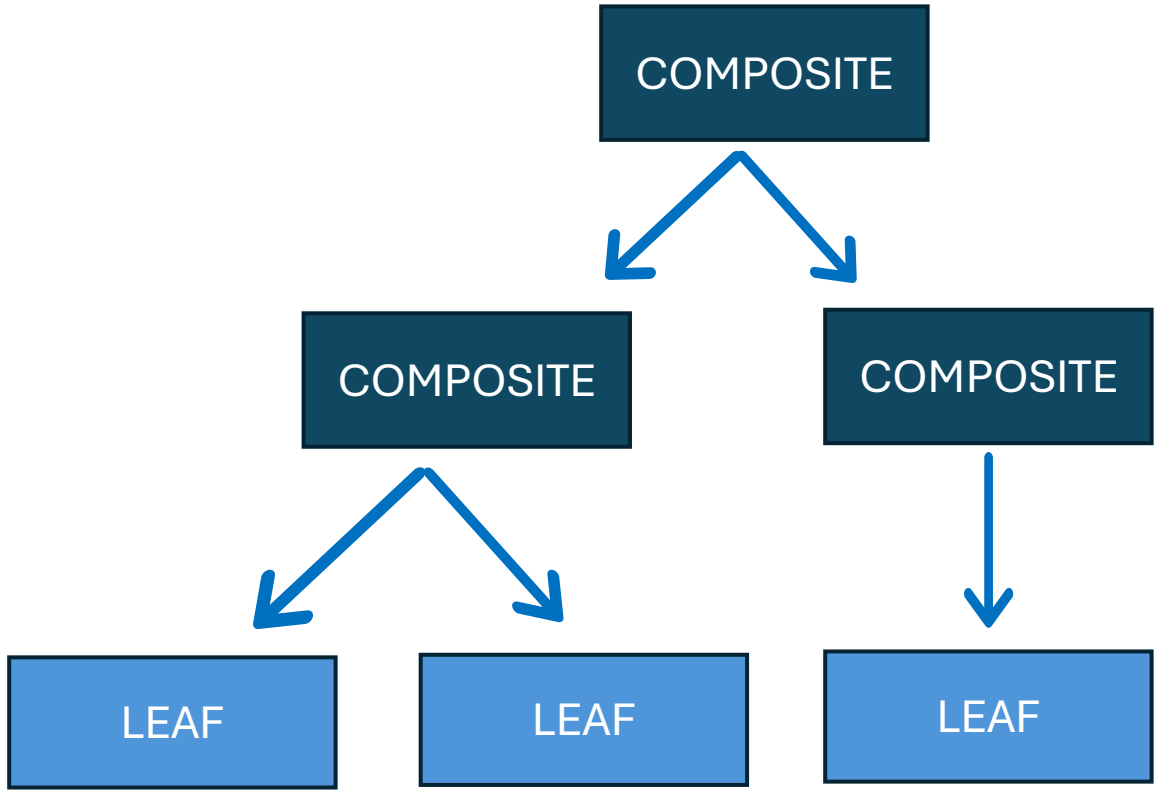
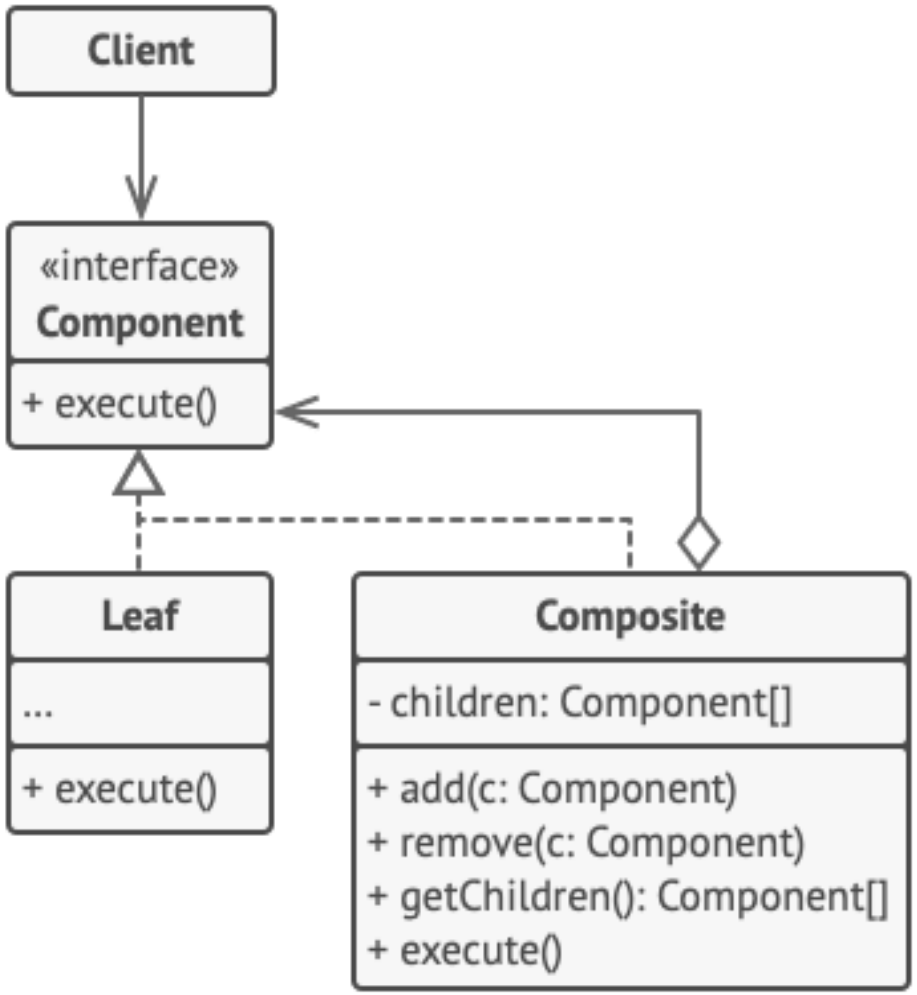
COMPOSITE

Composite is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects.

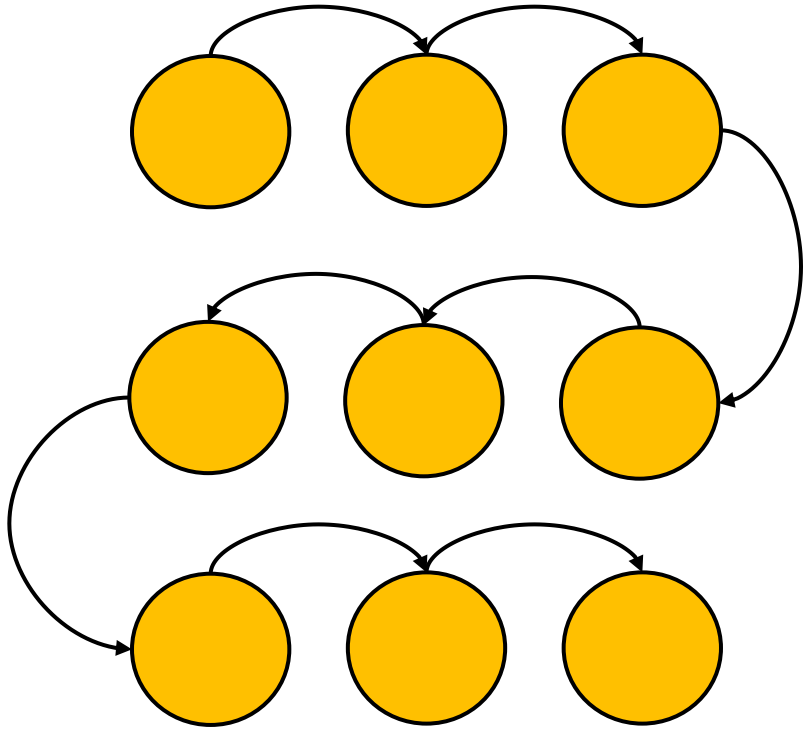


COMPOSITE

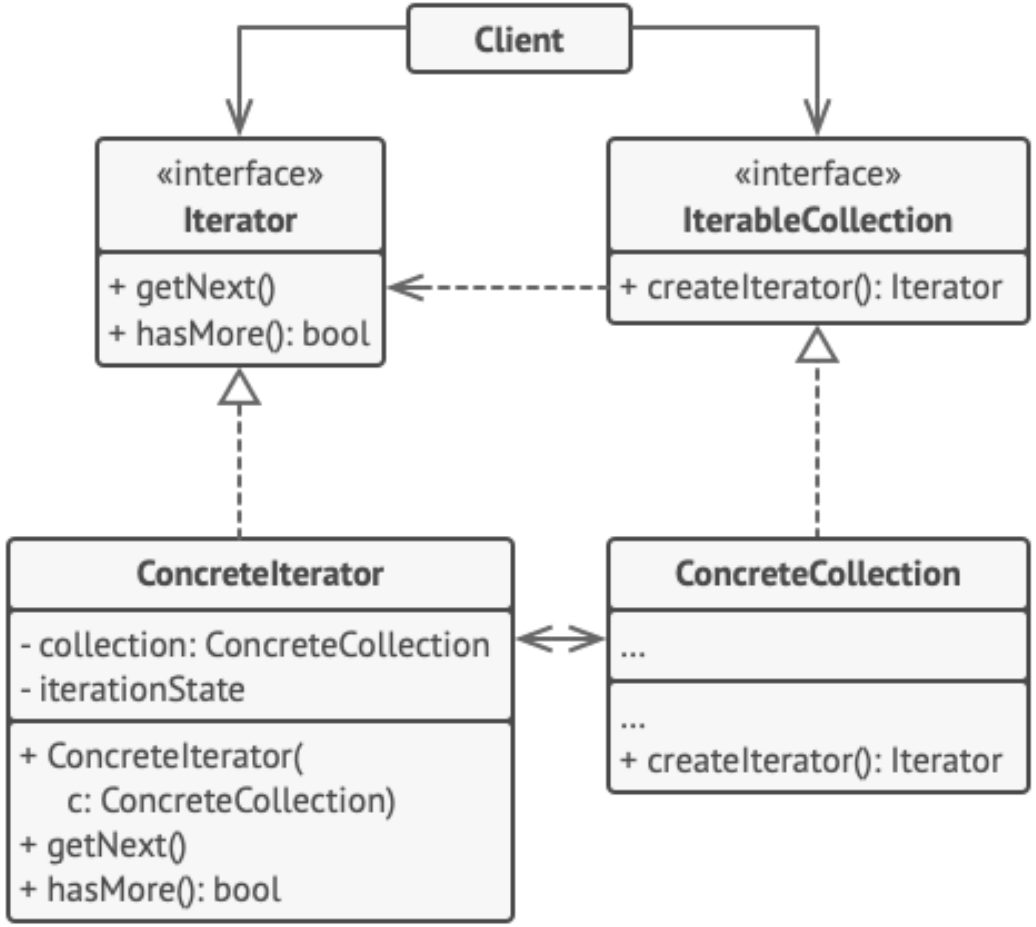

Client
«interface»
Component
+ execute()
Leaf
...
+ execute()
Composite
- children: Component[]
+ add(c: Component)
+ remove(c: Component)
+ getChildren(): Component[]
+ execute()

ITERATOR

Iterator is a behavioral design pattern that lets you traverse elements of a collection without exposing its underlying representation (list, stack, tree).



ITERATOR


Client
«interface»
Iterator
+ getNext()
+ hasMore(): bool
«interface»
IterableCollection
+ createIterator(): Iterator
ConcreteIterator
- collection: ConcreteCollection
- iterationState
+ ConcreteIterator(
c: ConcreteCollection)
+ getNext()
+ hasMore(): bool
ConcreteCollection
...
...
+ createIterator(): Iterator

ITERABLE AND ITERATOR

```
public interface Iterable<T> {
    Iterator<T> iterator();
}
```

```
public interface Iterator<E> {
    default void forEachRemaining(Consumer<? super E> action);
    boolean hasNext();
    E next();
    default void remove();
}
```

ITERATOR VS LIST ITERATOR

```
public interface ListIterator<E> extends Iterator<E> {
    boolean hasNext();
    E next();
    void remove();

    boolean hasPrevious();
    E previous();
    int nextIndex();
    int previousIndex();
    void set(E e);
    void add(E e);
}
```