XML-Generator

Fluid Interface és Composite patternek

Fluid Interface

Definíció:

A fluid interface minta lehetővé teszi a metódusláncolást, amely tisztább, olvashatóbb és intuitívabb kódot eredményez azáltal, hogy minden metódus a saját objektumát adja vissza.

Fő jellemzők:

- A metódusok *this*-t (az aktuális objektumot) adnak vissza, így láncolhatók.
- Elősegíti az immutabilitást konfigurációk vagy módosítások során.
- Javítja a fejlesztői produktivitást és a kód olvashatóságát.

Implementáció és használat

```
// Add a child node
add(child) {
  this.children.push(child)
  return this
}

// Remove a child node
remove(child) {
  this.children = this.children.filter(e => e !== child)
  return this
}
```

```
// Mutator for name
setName(name) {
  this.name = name
  return this
}
```

```
this.state = {
  root: new Composite().setName('SubComposite 1') // Create the root Composite node
    .add(
      (new Composite().setName('SubComposite 1'))
        .add((new Leaf()).setText('Leaf 1 Text').setName('Leaf1'))
    )
    .add((new Leaf()).setText('Leaf 2 Text').setName('Leaf2'))
};
```

Composite

Definíció:

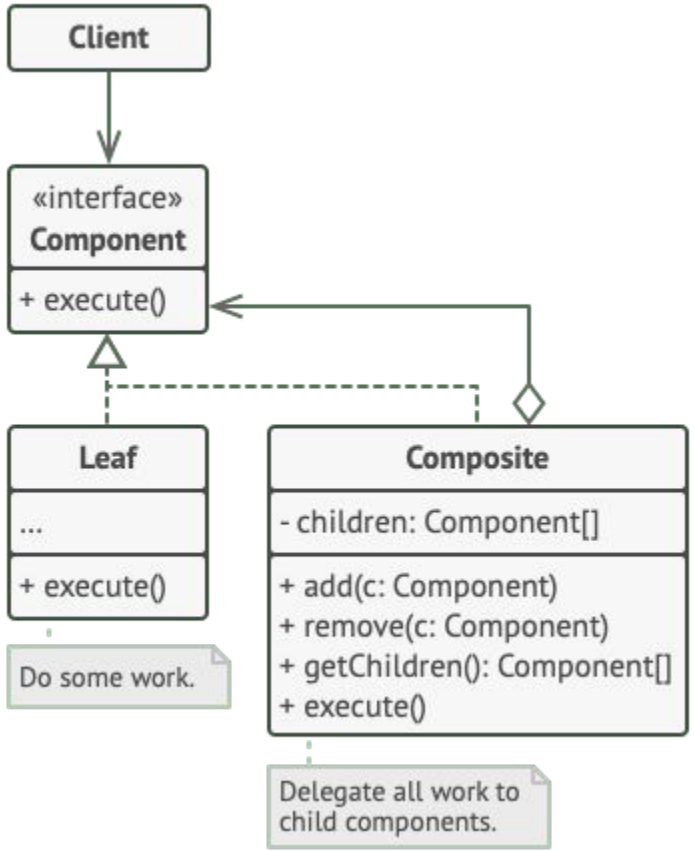
A composite minta lehetővé teszi, hogy az egyedi objektumokat és az objektumok kompozícióit egyenlően kezeljük. Ez egy fa struktúrát reprezentál, ahol az egyedi objektumok (levelek) és az objektumok csoportjai (kompozitok) egy közös interfészt valósítanak meg.

Fő jellemzők:

- A kompozitok más kompozitokat vagy leveleket tartalmazhatnak.
- Hierarchikus struktúrák egységes kezelése.
- Az operációkat rekurzívan alkalmazza, egyszerűsítve az összetett rendszerek kezelését.

Struktúra:

- Component: Absztrakt osztály vagy interfész, amelyet a levelek és kompozitok megosztanak.
- Leaf: Alap építőelem, amely nem tartalmaz gyermekeket.
- Composite: Tartalmazhat gyermekeket (leveleket vagy más kompozitokat).



Implementáció és használat

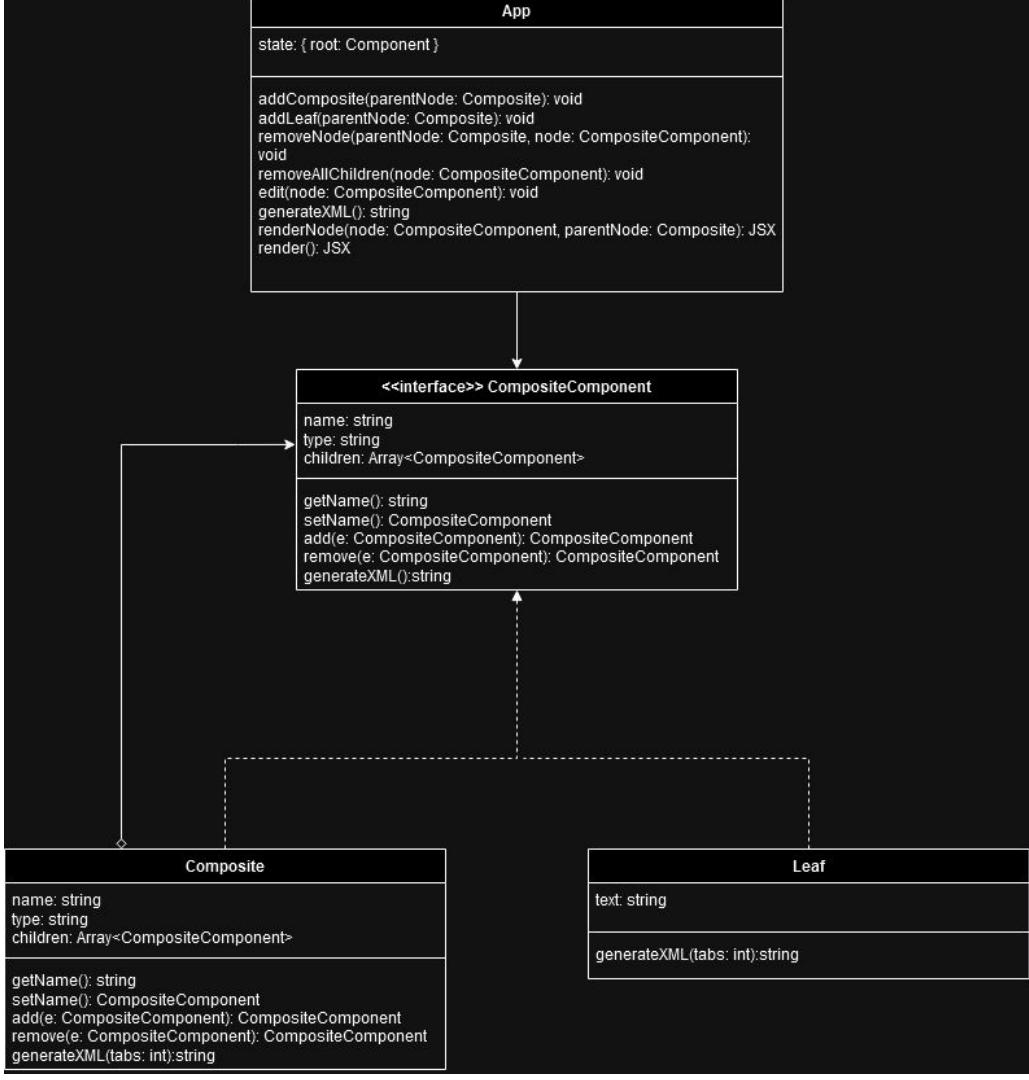
```
// Add a child node
add(child) {
  this.children.push(child)
  return this
}

// Remove a child node
remove(child) {
  this.children = this.children.filter(e => e !== child)
  return this
}
```

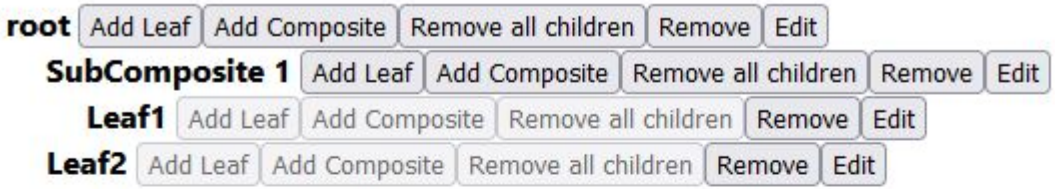
```
// Mutator for name
setName(name) {
  this.name = name
  return this
}
```

```
this.state = {
  root: new Composite().setName('SubComposite 1') // Create the root Composite node
    .add(
      (new Composite().setName('SubComposite 1'))
        .add((new Leaf()).setText('Leaf 1 Text').setName('Leaf1'))
    )
    .add((new Leaf()).setText('Leaf 2 Text').setName('Leaf2'))
};
```

UML


App
state: { root: Component }
addComposite(parentNode: Composite): void
addLeaf(parentNode: Composite): void
removeNode(parentNode: Composite, node: CompositeComponent): void
removeAllChildren(node: CompositeComponent): void
edit(node: CompositeComponent): void
generateXML(): string
renderNode(node: CompositeComponent, parentNode: Composite): JSX
render(): JSX
<<interface>> CompositeComponent
name: string
type: string
children: Array<CompositeComponent>
getName(): string
setName(): CompositeComponent
add(e: CompositeComponent): CompositeComponent
remove(e: CompositeComponent): CompositeComponent
generateXML():string
Composite
name: string
type: string
children: Array<CompositeComponent>
getName(): string
setName(): CompositeComponent
add(e: CompositeComponent): CompositeComponent
remove(e: CompositeComponent): CompositeComponent
generateXML(tabs: int):string
Leaf
text: string
generateXML(tabs: int):string

Demoszerűség



Generated XML:

```
<root>
        <SubComposite 1>
                <Leaf1>
                        Leaf 1 Text
                </Leaf1>
        </SubComposite 1>
        <Leaf2>
                Leaf 2 Text
        </Leaf2>
 </root>
```

KÖSZÖNÖM A FIGYELMET!