* Composite design pattern are 2 roluri:

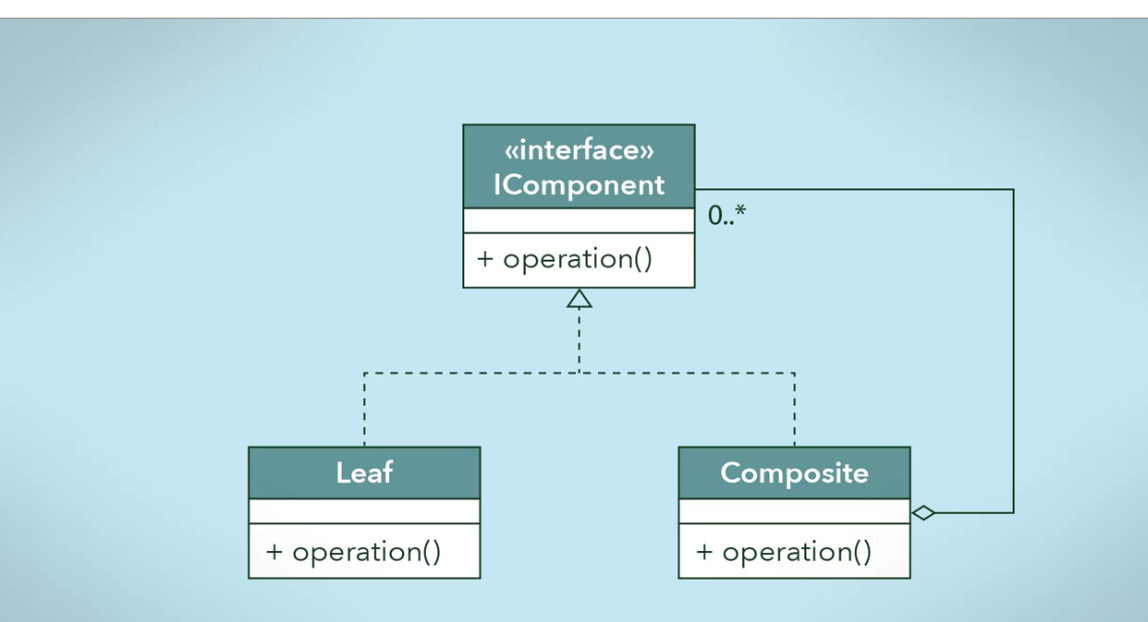
1. De a compune un obiect ce lucreaza cu o colectie de obiecte
2. Sa trateze aceste obiecte in mod identic.

* Composite foloseste **Inheritance** si **Composition**
* Composite design pattern e format din urmatoarele elemente:

- **Component** – interfata sau clasa abstracta pe care leaf si composite trebuie sa o implementeze.

- **Leaf** – sunt elementele din Composite, care implementeaza si ele component, si sunt obiecte independente. Un leaf nu are in el inca un composite sau component

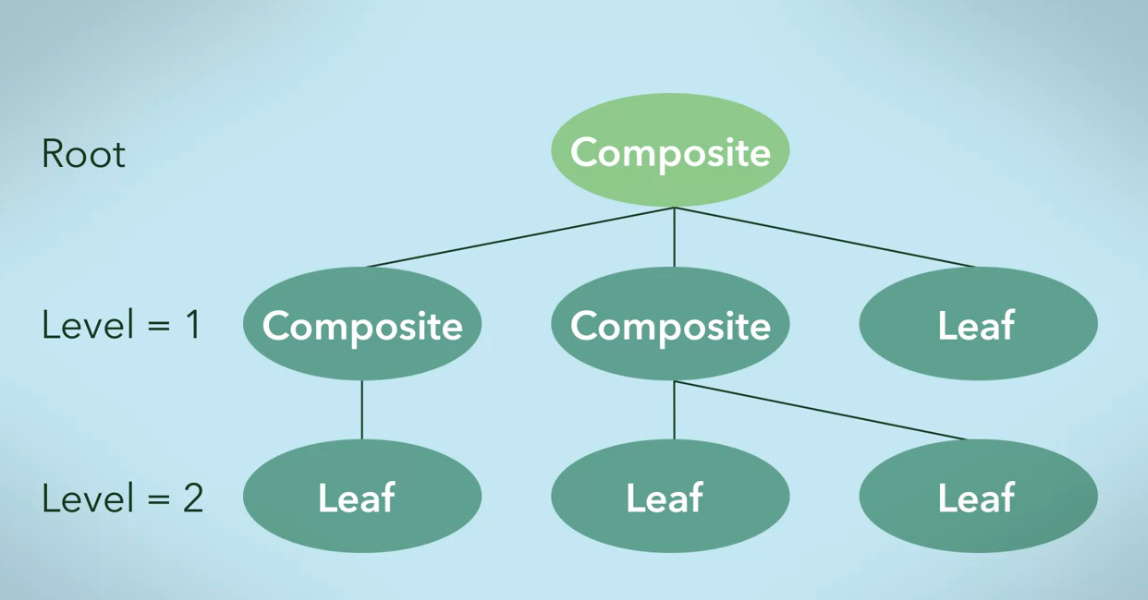
- **Composite** – clasa ce implementeaza si ea Component, dar lucreaza cu mai multe leaves sau composite, identic cu el, folosindu-le pe toate in metodele mostenite de la interfata executand in fiecare metoda mostentia aceeasi metoda pentru fiecare leaf sau composite



vedem ca Composite foloseste Composition dar si inheritance, caci va avea o List<> de Leaf sau Composite(clasele ce implementeaza Component interface)

Leaf foloseste doar inheritance

* Un Composite object poate contine si alte Composite objects, si asta se numeste **Recursive Composition**

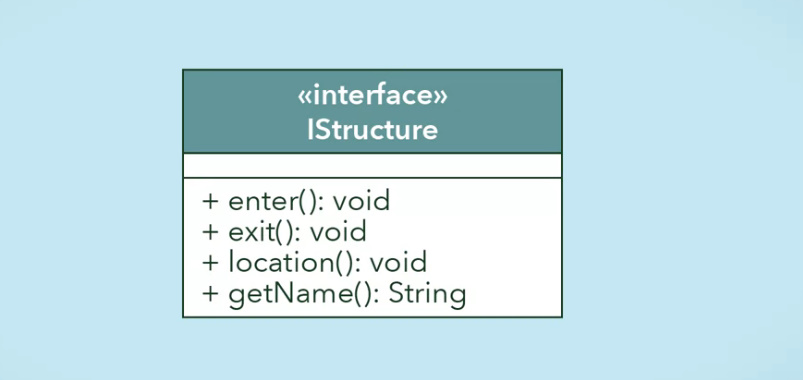


Un Composite va avea o List<> de alte Composite si fiecare Composite inca o List<> de Leaf

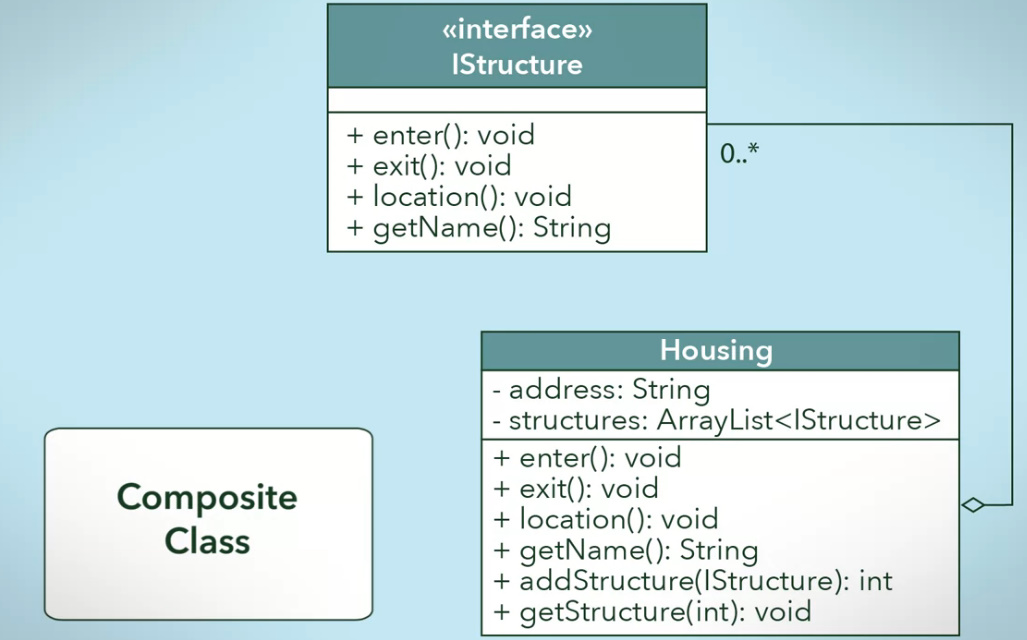
* Din cauza la nautura ciclica, e considerat si ca un Tree
* Composite rezolva 2 probleme:

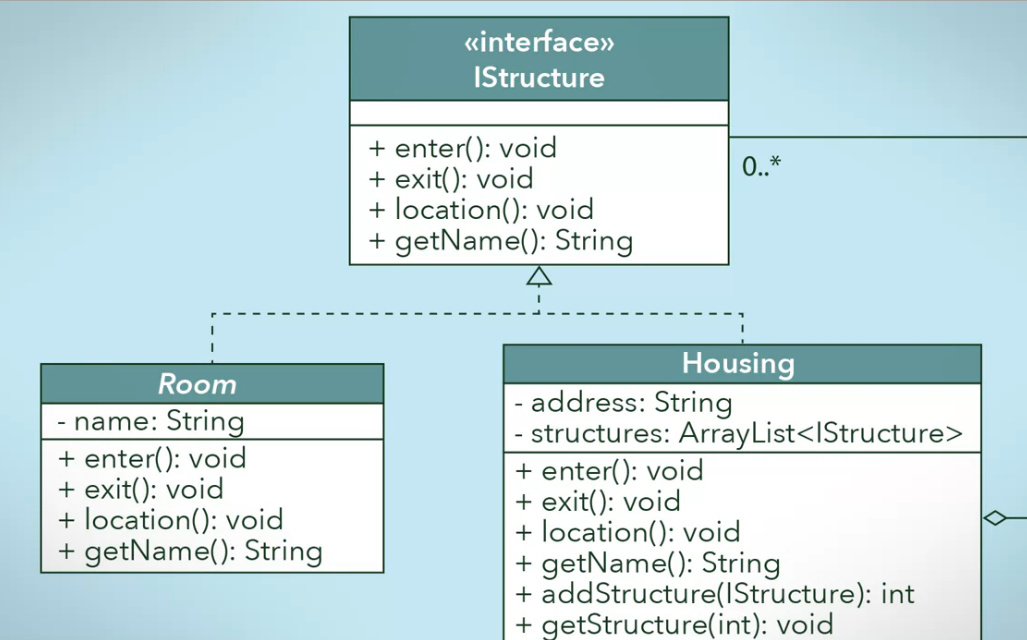
1. Crearea unui tree ce contine diferite tipuri de obiecte, dar cu un parinte comun
2. Crearea unei clase ce poate lucra cu obiecte difertie ca tip, dar ca parinte comun, si care sa le trateze pe toate identic.

**Exemplu I**

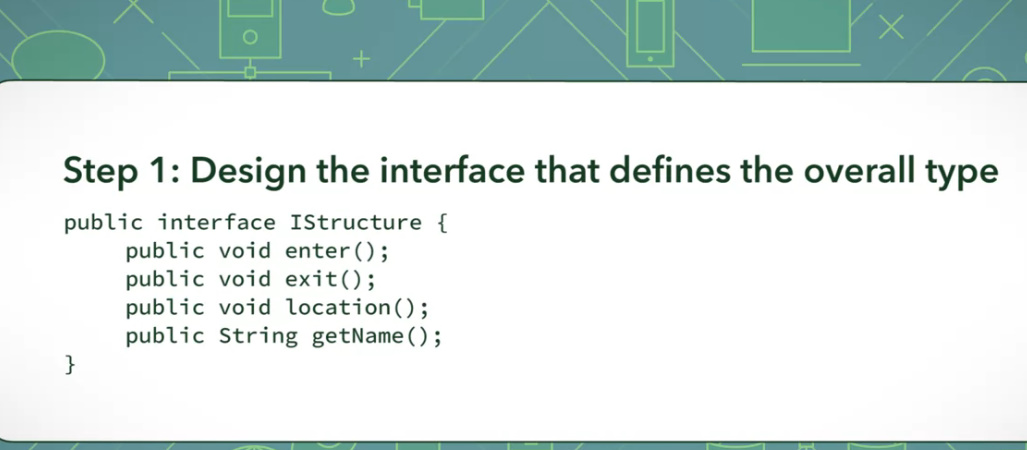


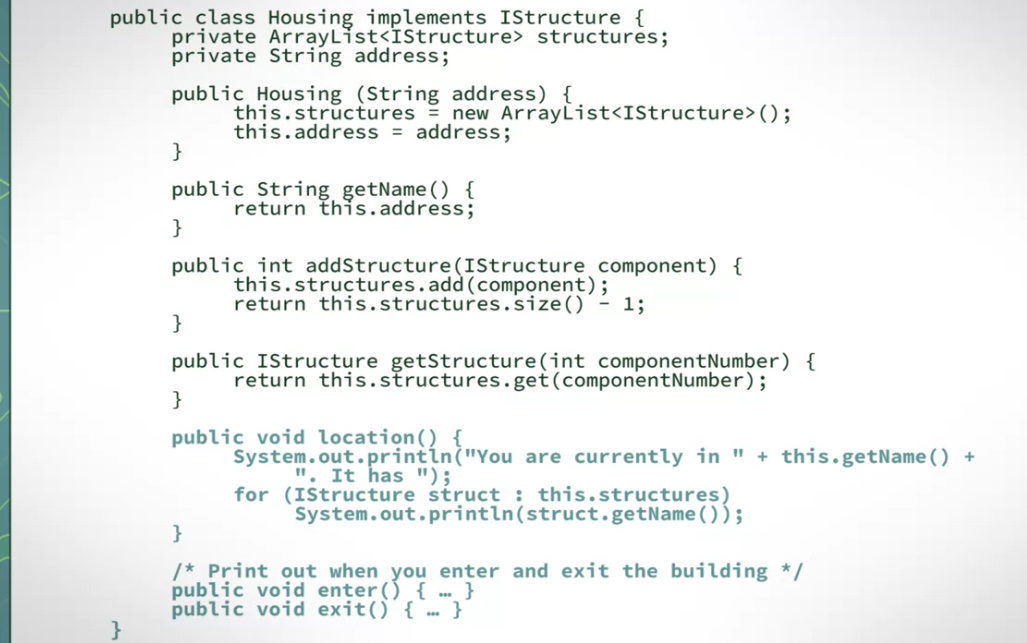
Structure se refera la orice parte dintr-o casa, ca room, floor etc.



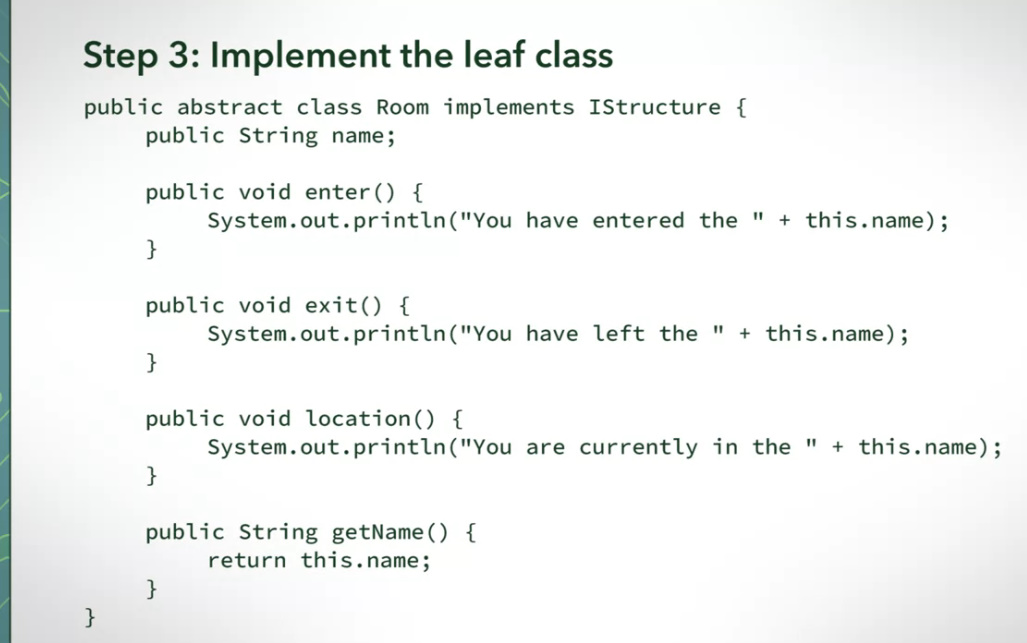


Housing poate contine in ArrayList<> atat Room objects si tot Housing objects, adica identice cu el.





in enter() si exit() vom apela tot enter() si exit() pentru fiecare object din ArrayList, **dar nu e obligaatoriu ca in metoda mostenita de la interfata, sa apelam metoda cu acelasi nume pentru toate elementele din list.**





Putem avea si alte implementari ale lui Istructure ca Leaf pe langa Room, si deobicei si vom avea.

**Exemplu II**

**Component**

public interface Shape {

public void draw(String fillColor);

}

**Leaves:**

public class Triangle implements Shape {

@Override

public void draw(String fillColor) {

System.out.println("Drawing Triangle with color "+fillColor);

}

}

public class Circle implements Shape {

@Override

public void draw(String fillColor) {

System.out.println("Drawing Circle with color "+fillColor);

}

}

**Composite**

public class Drawing implements Shape{

//collection of Shapes

private List<Shape> shapes = new ArrayList<Shape>();

@Override

public void draw(String fillColor) {

for(Shape sh : shapes)

{

sh.draw(fillColor);

}

}

//adding shape to drawing

public void add(Shape s){

this.shapes.add(s);

}

//removing shape from drawing

public void remove(Shape s){

shapes.remove(s);

}

//removing all the shapes

public void clear(){

System.out.println("Clearing all the shapes from drawing");

this.shapes.clear();

}

}

Composite a folosid metoda draw(), si in ea a folosit iar metoda draw() pentru fiecare element din ArrayList<>.

public class TestCompositePattern {

public static void main(String[] args) {

Shape tri = new Triangle();

Shape tri1 = new Triangle();

Shape cir = new Circle();

Drawing drawing = new Drawing();

drawing.add(tri1);

drawing.add(tri1);

drawing.add(cir);

drawing.draw("Red");

drawing.clear();

drawing.add(tri);

drawing.add(cir);

drawing.draw("Green");

}

}