Design Patterns

Singleton, Composite

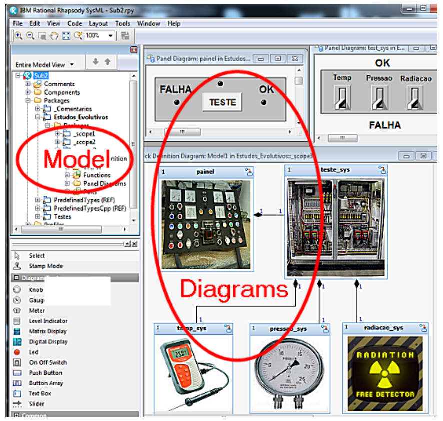
COMP3607
Object Oriented Programming II

12-Sept-2018

Outline

- Drawing vs Modelling
- Singleton Design Pattern
- Composite Design Pattern

Drawing vs Modelling



https://www.ibm.com/developerworks/community/blogs/invisiblethread/entry/a_diagram_is_not_a_model_the_huge_difference_between_them?lang=en_us

1 Singleton Design Pattern

The Singleton design pattern is used to ensure that a class has only one instance, and to provide a global point of access to it.

Reading Resource: Design Patterns, Chapter 3, pp 127-134

Problem Scenario

Consider a scenario where a system has many printers, but there should be only one printer spooler*.

How do we ensure that the printer spooler class has only **one** instance of a printer spooler object, and that the instance is **easily accessible**?

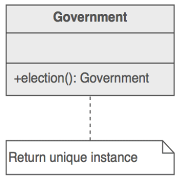
Possible options:

- A global variable - problem with this?
- Make the class responsible for keeping track of the sole instance and for giving access to it.

* software responsible for managing all print jobs by adding and removing jobs from a queue.

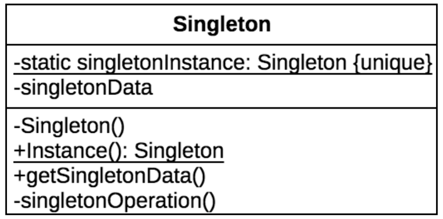
Another Example

The Singleton pattern ensures that a class has only one instance and provides a global point of access to that instance. It is named after the singleton set, which is defined to be a set containing one element. The office of the President of the United States is a Singleton. The United States Constitution specifies the means by which a president is elected, limits the term of office, and defines the order of succession. As a result, there can be at most one active president at any given time. Regardless of the personal identity of the active president, the title, "The President of the United States" is a global point of access that identifies the person in the office.



https://sourcemaking.com/design_patterns/singleton

Singleton Design Pattern (UML)



Reading Resource: Chapter 3: Design Patterns

Code Example

```
public class Singleton {

    private static Singleton singletonInstance;//unique instance
    private String singletonData;

    private Singleton() {// private constructor
       singletonData = “A single tonne of data is a lot of data”;
    }

    public static Singleton Instance() { //class method to get to unique instance
      if(singletonInstance == null)
          singletonInstance = new Singleton();
        return singletonInstance;
    }

    public String getSingletonData() {
        return singletonData;
    }

    private void singletonOperation() {
        // TODO implement other stuff here
    }

}
```

Applicability: Singleton

The Singleton pattern is used when:

- There must be exactly one instance of class, and it must be accessible to clients from well-known access point.
- The sole instance should be extensible by subclassing, and clients should be able to use an extended instance without modifying their code.

1

Consequences: Singleton

Benefits:

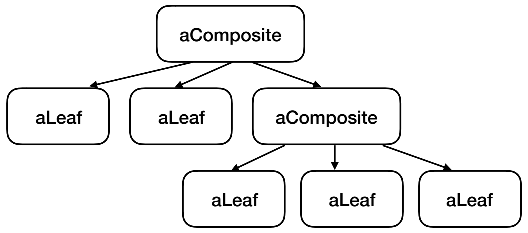
- Controlled access to sole instance: (encapsulation)
- Reduced name space: (less global variables)
- Refinement of operations: (subclassing)
- Variable number of instances: (controlled by class)
- More flexible than class operations: (language quirks)

1

Composite Design Pattern

The Composite design pattern is used to compose objects into tree structures to represent part-whole hierarchies.

It lets clients treat individual objects and compositions of objects uniformly.



2

Problem Scenario

Suppose we are building a graphics application (drawing editor) to let users build complex diagrams out of simple components.

Users can group components to make larger ones which can be grouped further to make still larger ones and so on.

How can we achieve this?

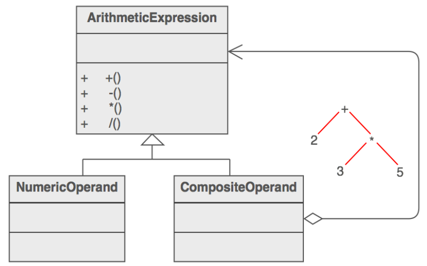
Possible options:

- Simple implementation: Use classes for Graphical Primitives (Lines, Text) and other container classes for these primitives. Problem?
- An abstract class that represents both primitives and their containers.

2

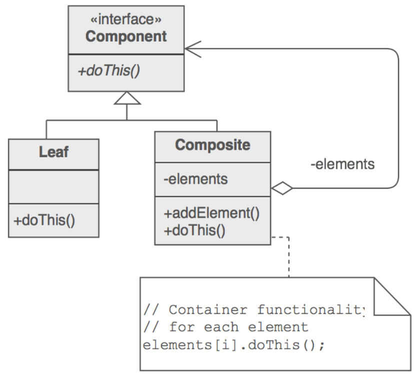
Another Example

The Composite composes objects into tree structures and lets clients treat individual objects and compositions uniformly. Although the example is abstract, arithmetic expressions are Composites. An arithmetic expression consists of an operand, an operator (+ - * /), and another operand. The operand can be a number, or another arithmetic expression. Thus, 2 + 3 and (2 + 3) + (4 * 6) are both valid expressions.



https://sourcemaking.com/design_patterns/composite

Composite Design Pattern (UML)



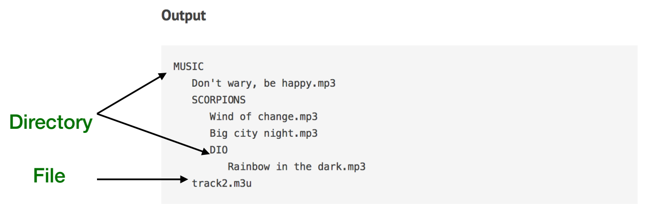
```
// Container functionalit
// for each element
elements[i].doThis();
```

Code Example

```
public interface Component{
     public abstract void doThis(); //common operation in hierarchy
}
public class Composite implements Component{
     private Collection <Component> toDo; //initialised in constructor
     public void addComponent(Component c){
        //code to add ‘c’ to the Collection
     }
     public void doThis(){
        //do something useful with the components in Collection (iterate)
     }
}
public class Leaf implements Component{
    public void doThis(){ //something happens here in Leaf
     }
}
```

Exercise

Design a solution for listing all of the files and directories stored on a disk using the Composite Design Pattern. Output should be presented as:



Output

```
MUSIC
  Don't wary, be happy.mp3
  SCORPIONS
    Wind of change.mp3
    Big city night.mp3
    DIO
      Rainbow in the dark.mp3
  track2.m3u
```

https://sourcemaking.com/design_patterns/composite/java/1

Applicability: Composite Pattern

The Composite pattern is used 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.

2

Consequences: Composite Pattern

Benefits and Liabilities

- Defines class hierarchies consisting of leaf and composite objects. Complex objects can be composed of leaf objects recursively.
- Makes clients simple: they can treat composite and leaf objects uniformly.
- Easier to add new kinds of components (as leaf or composite classes)
- Harder to restrict components of a composite object using types; run-time checks needed instead.

2

References

- UML: online reading resources
  - www.uml.org (Documentation, notation)
  - www.omg.org (Standards, protocols)
- UML Modelling and Drawing Tool:
  - http://staruml.io/