COMP2511

Week 7

TUESDAY 9AM - 12PM (T09B)

TUESDAY 1PM - 4PM (T13B)

This week

- Composite pattern
- Factory pattern

Vivas for assignment-i to be finished this week

Assignment Feedback

Assignment Feedback

- Use `.equal()` for string comparison instead of `==`
- Avoid the use of magic numbers, assign them to final variables (readability)
- Don't use `super.x` to set attributes in the super class in the subclass's constructor. Pass variable into super constructor as argument
- Use `instanceof` for type comparison
- Polymorphism is preferred over typechecking to perform a specific action
- Format your code for consistent whitespace

Assignment Feedback

```java
public class Satellite {
    private String name;
    protected double range;

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

    public double getRange() {
        return this.range;
    }
}

public class StandardSatellite extends Satellite {
    public StandardSatellite(String name, double range) {
        super(name);
        super.range = range; // don't do this
        // super(name, range); // do this
    }

    public static void main(String[] args) {
        Satellite s = new Satellite("hello");
        s.getRange(); // Potentially undefined behaviour
    }
}
```

Group Task

Finding patterns

Finding Pattern

In groups, determine a possible pattern that could be used to solve each of the following problems:

1. Sorting collections of records in different orders.
   - Strategy
2. Modelling a file system
   - Composite
3. Updating a UI component when the state of a program changes
   - Observer
4. Parsing and evaluating arithmetic expressions
   - Composite
5. Adjusting the brightness of a screen based on a light sensitivity
   - Observer

Composite Pattern

Composite Pattern

What type of design pattern is composite?

Structural

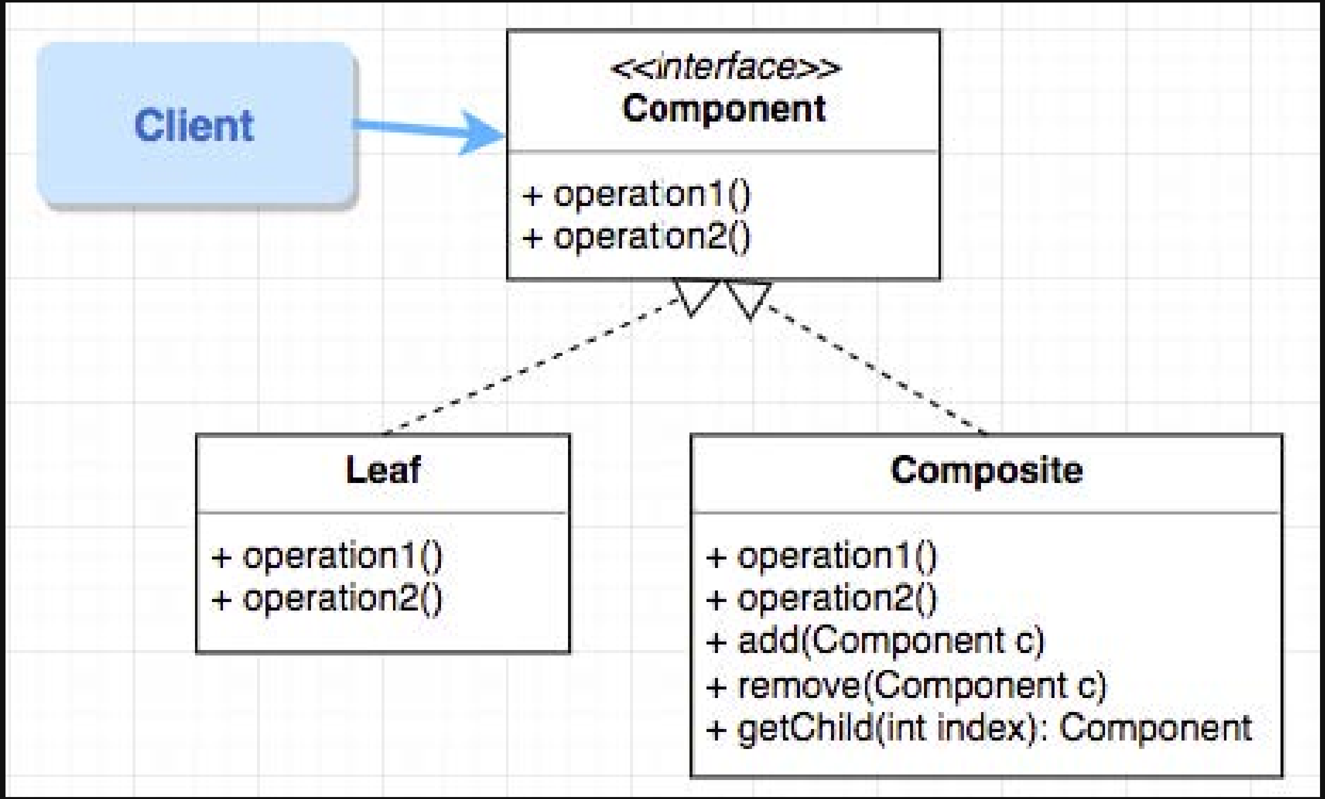
Structural design pattern are patterns that ease the design by identifying a simple way to realize relationships among entities.

They explain how to assemble objects and classes into large structures, while keeping structures flexible and efficient

Composite pattern is useful for aggregating different objects/data. The aim is to be able to manipulate a single instance of an object just as you would manipulate a group of them.

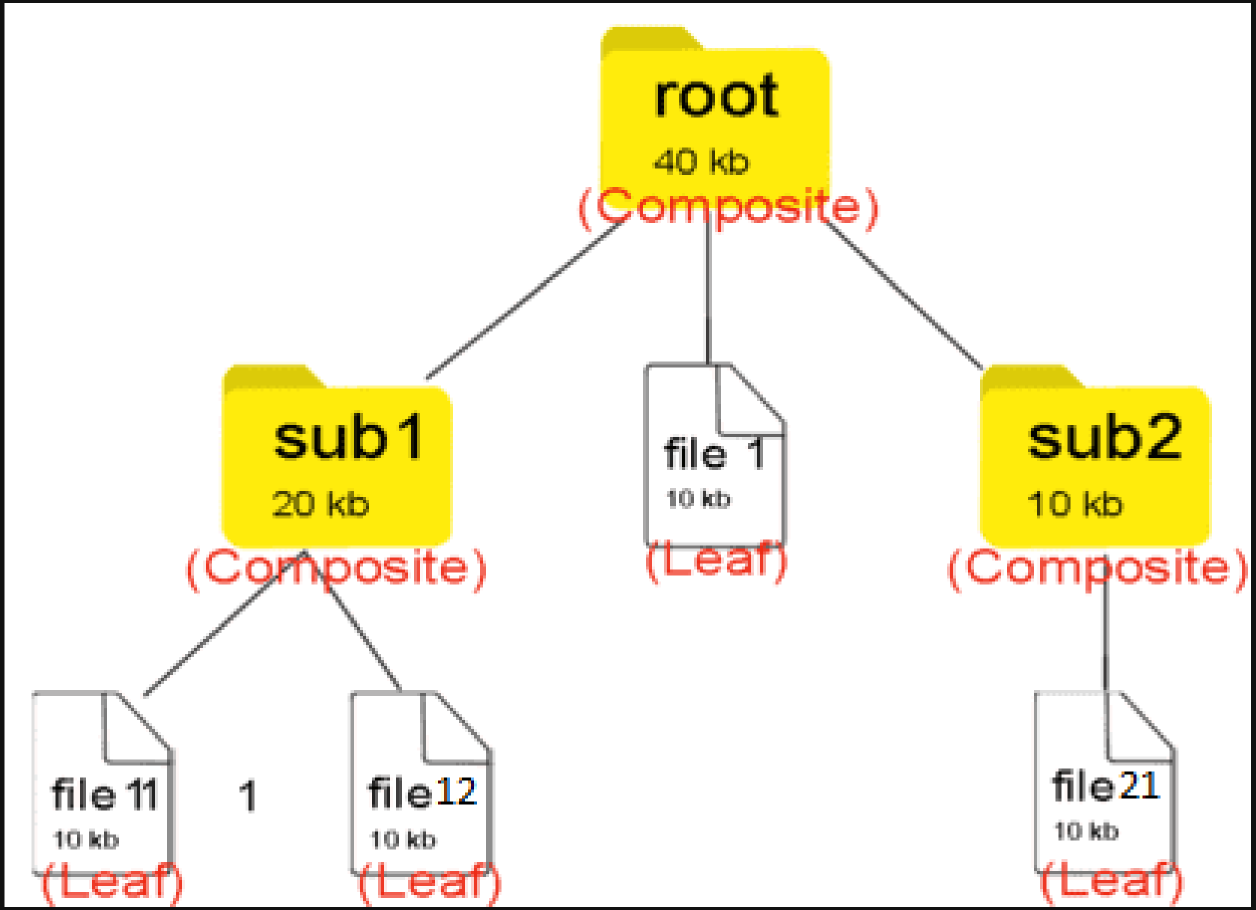
Tree like structure of objects

Composite Pattern



- No discrimination between a single (leaf) or a group (composite). Keeps code clean

Composite Pattern



Code Demo

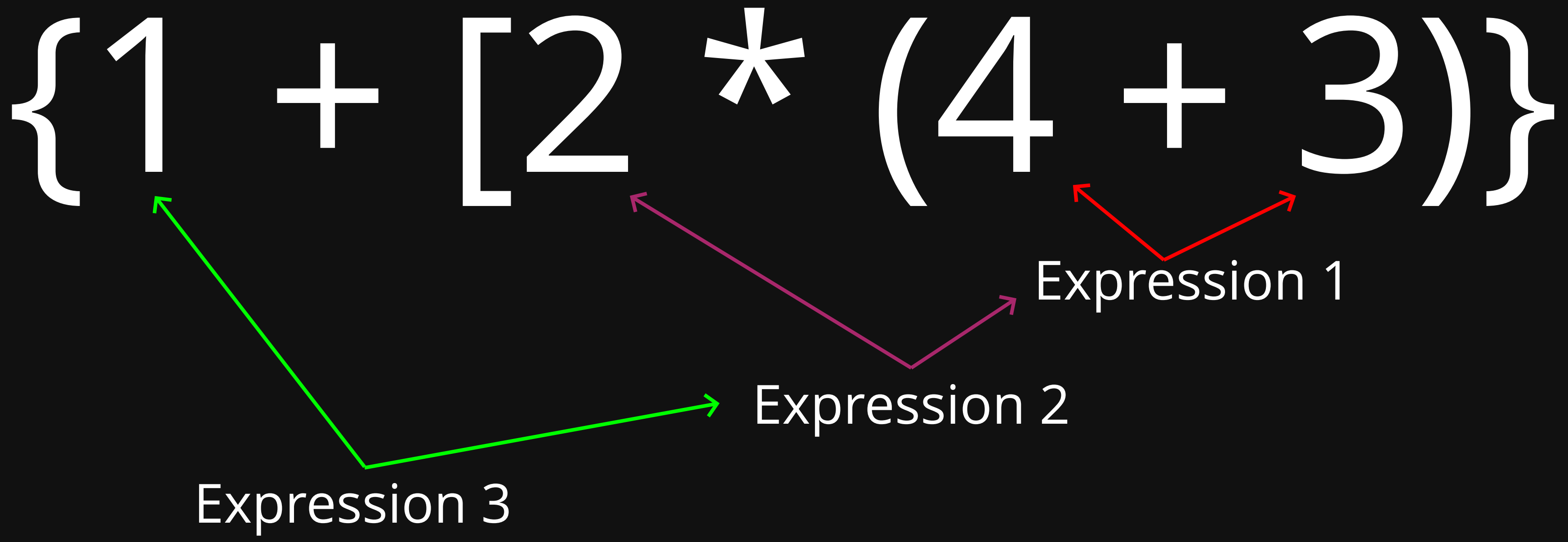
Calculator.java - Composite pattern

Code Demo

Inside **`src/calculator`**, use the Composite Pattern to write a simple calculator that evaluates an expression. Your calculator should be able to:

- Add two expressions
- Subtract two expressions
- Multiply two expressions
- Divide two expressions

Code Demo



Factory Pattern

Factory Pattern

What type of design pattern?

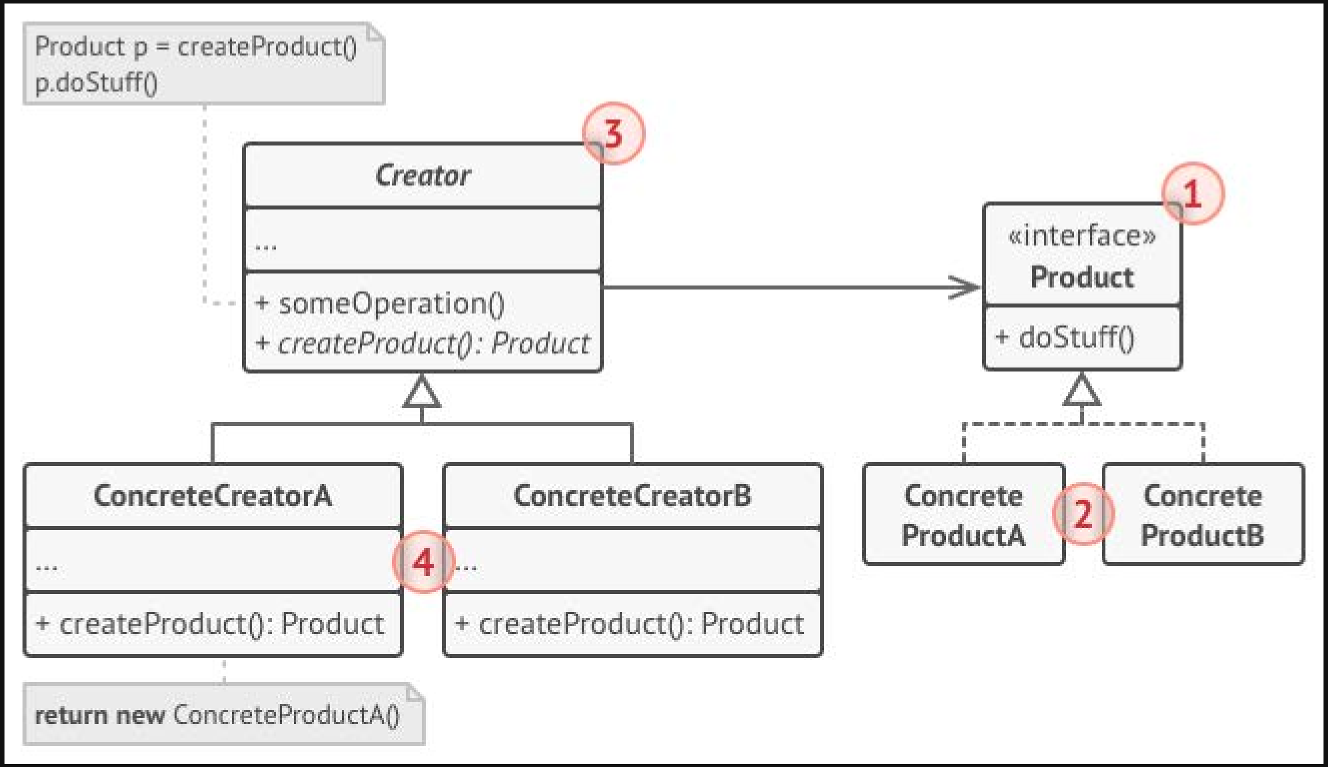
Creational

Creational patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

Factory method provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.

Reduces coupling and shotgun surgery, as all classes are created using the same method.

Factory Pattern



1. The **Product** declares the interface, which is common to all objects that can be produced by the **Creator** and its subclasses.
2. **Concrete products** are different implementations of the product interface
3. The **Creator** class declares the factory method and returns new product objects
4. **Concrete Creators** override the base factory method so it returns a new type of product

Code Demo

Thrones.java - Factory Pattern

Code Demo

Inside `src/thrones`, there is some code to model a simple chess-like game. In this game different types of characters move around on a grid fighting each other. When one character moves into the square occupied by another they attack that character and inflict damage based on random chance. There are four types of characters:

- A king can move one square in any direction (including diagonally), and always causes 8 points of damage when attacking.
- A knight can move like a knight in chess (in an L shape), and has a 1 in 2 chance of inflicting 10 points of damage when attacking.
- A queen can move to any square in the same column, row or diagonal as she is currently on, and has a 1 in 3 chance of inflicting 12 points of damage or a 2 out of 3 chance of inflicting 6 points of damage.
- A troll can only move up, down, left or right, and has a 1 in 6 chance of inflicting 20 points of damage.

Code Demo

We want to refactor the code so that when the characters are created, they are put in a random location in a grid of length 5.

1. How does the Factory Pattern (AKA Factory Method) allow us to abstract construction of objects, and how will it improve our design with this new requirement?
    - Abstract the construction of the character objects. We don't deal with the constructor, instead call a general factory method that handles the number.
2. Use the Factory Pattern to create a series of object factories for each of the character types, and change the `main` method of `Game.java` to use these factories.

Attendance

Feedback

https://forms.gle/R4sMTTQzPC4vqXSN8