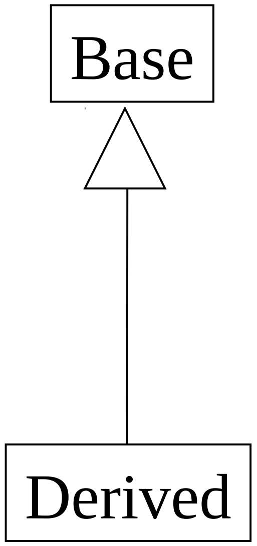
VIETNAM NATIONAL UNIVERSITY - HCM
International Education Institute

Pre-OOP Course

Giới thiệu về Java

Lecturer: Dr. Nguyễn Minh Hải
(Ho Chi Minh University of Technology)

Inheritance



```
class Base {
    Base(){}
    Base(int i) {}
    protected void foo() {…}
}

class Derived extends Base {
    Derived() {}
    protected void foo() {…}
    Derived(int i) {
   super(i);
       …
      super.foo();
    }
}
```

As opposed to C++, it is possible to inherit only from ONE class.

Pros avoids many potential problems and bugs.

Cons might cause code replication

Inheritance (2)

```
class Base {
  void foo() {
    System.out.println(“Base”);
  }
}
class Derived extends Base {
  void foo() {
    System.out.println(“Derived”);
  }
}
public class Test {
  public static void main(String[] args) {
    Base b = new Derived();
    b.foo();  // Derived.foo() will be activated
  }
}
```

Inheritance (3) - Optional

```
class classC extends classB {
   classC(int arg1, int arg2){
     this(arg1);
     System.out.println("In classC(int arg1, int arg2)");
   }
   classC(int arg1){
     super(arg1);
     System.out.println("In classC(int arg1)");
   }
}
class classB extends classA {
   classB(int arg1){
     super(arg1);
     System.out.println("In classB(int arg1)");
   }
   classB(){
     System.out.println("In classB()");
   }
}
```

Inheritance (3) - Optional

```
class classA {
   classA(int arg1){
    System.out.println("In classA(int arg1)");
   }
   classA(){
     System.out.println("In classA()");
   }
}

class classB extends classA {
   classB(int arg1, int arg2){
     this(arg1);
     System.out.println("In classB(int arg1, int arg2)");
   }
   classB(int arg1){
     super(arg1);
     System.out.println("In classB(int arg1)");
   }

  class B() {
    System.out.println("In classB()");
  }
}
```

Polymorphism

- Inheritance creates an “is a” relation:

For example, if B inherits from A, than we say that “B is also an A”.

Implications are:

- access rights (Java forbids reducing access rights) - derived class can receive all the messages that the base class can.
- behavior
- precondition and postcondition

Abstract

- ***abstract*** member function, means that the function does not have an implementation.
- ***abstract*** class, is class that can not be instantiated.

```
AbstractTest.java:6: class AbstractTest is an abstract class.
It can't be instantiated.
        new AbstractTest();
        ^
1 error
```

NOTE:
An abstract class is not required to have an abstract method in it.
But any class that has an abstract method in it or that does
not provide an implementation for any abstract methods declared
in its superclasses must be declared as an abstract class.

Example

Abstract - Example

```
package java.lang;
public abstract class Shape {
    public abstract void draw();
    public void move(int x, int y) {
      setColor(BackGroundColor);
          draw();
          setCenter(x,y);
      setColor(ForeGroundColor);
          draw();
        }
}
```

```
package java.lang;
public class Circle extends Shape {
    public void draw() {
          // draw the circle ...
        }
}
```

Interface

Interfaces are useful for the following:

- Capturing similarities among unrelated classes without artificially forcing a class relationship.
- Declaring methods that one or more classes are expected to implement.
- Revealing an object's programming interface without revealing its class.

Interface

- abstract “class”
- Helps defining a “usage contract” between classes
- All methods are public
- Java’s compensation for removing the multiple inheritance. You can “inherit” as many interfaces as you want.

* - The correct term is “to implement” an interface

Example

Interface

```
interface IChef {
   void cook(Food food);
}
```

```
interface BabyKicker {
   void kickTheBaby(Baby);
}
```

```
interface SouthParkCharacter {
   void curse();
}
```

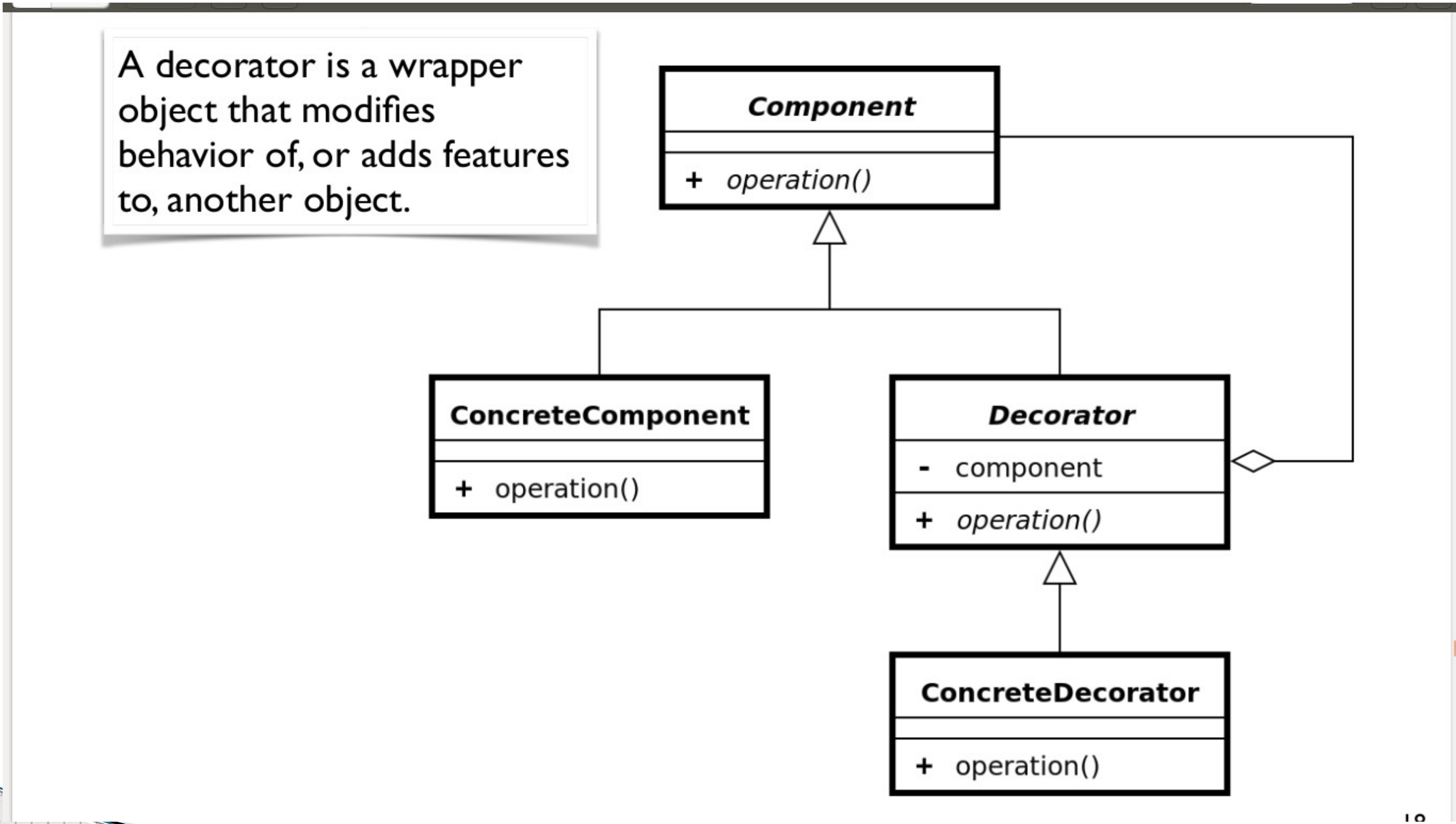
```
class Chef implements IChef, SouthParkCharacter {
   // overridden methods MUST be public
   // can you tell why ?
   public void curse() { … }
   public void cook(Food f) { … }
}
```

* access rights (Java forbids reducing of access rights)

When to use an interface ?

Perfect tool for encapsulating the classes inner structure. Only the interface will be exposed

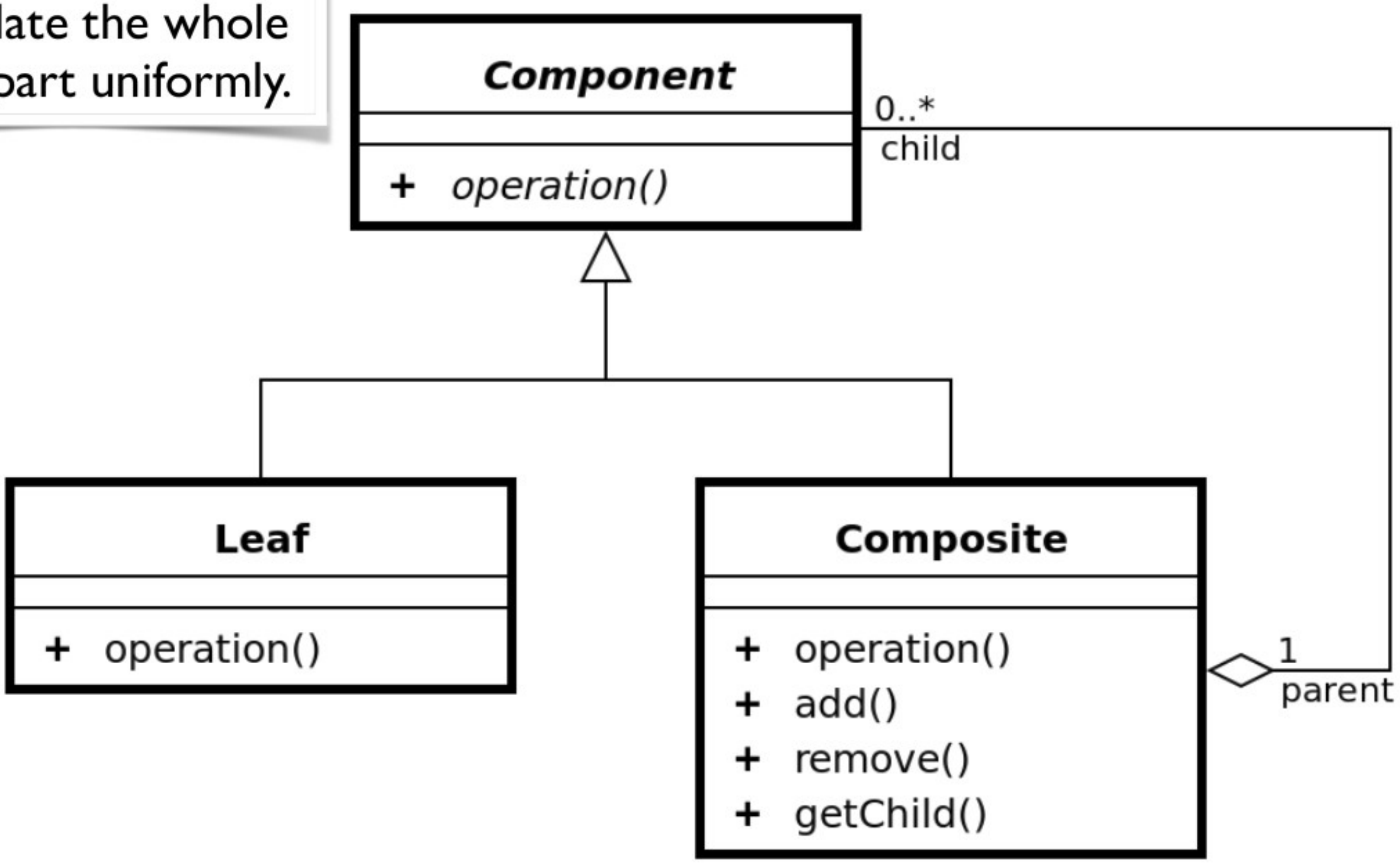
Decorator Pattern



A decorator is a wrapper object that modifies behavior of, or adds features to, another object.

Composite Pattern

A client can manipulate the whole or any part uniformly.



THANKS FOR LISTENING