**Composite**

**Definition**

Build a complex object out of elemental objects and itself like a tree structure.

**Where to use & benefits**

* Want to represent a part-whole relationship like tree folder system
* Group components to form larger components, which in turn can be grouped to form still larger components.
* Related patterns include
  + [Decorator](http://www.javacamp.org/designPattern/decorator.html), which is often used with composite pattern and with the same parent class.
  + [Flyweight](http://www.javacamp.org/designPattern/flyweight.html), which is used with composite pattern to share components.
  + [Iterator](http://www.javacamp.org/designPattern/iterator.html), which is used to traverse the composites.
  + [Visitor](http://www.javacamp.org/designPattern/visitor.html), which localizes operations across composite and leaf classes.

**Example**

A component has many elements and itself which has many elements and itself, etc. A file system is a typical example. Directory is a composite pattern. When you deal with Directory object, if isFile() returns true, work on file, if isDirectory() returns true, work on Directory object.

class Directory {

Directory dir;

File[] f;

...

boolean isDirectory() {

return f == null;

}

boolean isFile() {

return f != null;

}

File getFile(int i) {

if (isFile())

return f[i];

return null'

}

Directory getDirectory() {

if (isDirectory())

return dir;

return null;

}

....

}

For example, General Manager may have several employees and some of employees are Managers which have several employees. To illustrate such issue, we design a simple Manager class.

class Employee {

String name;

double salary;

Employee(String n, double s){

name = n;

salary = s;

}

String getName() {

return name;

}

double getSalary() {

return salary;

}

public String toString() {

return "Employee " + name;

}

}

class Manager {

Manager mgr;

Employee[] ely;

String dept;

Manager(Manager mgr,Employee[] e, String d ) {

this(e, d);

this.mgr = mgr;

}

Manager(Employee[] e, String d) {

ely = e;

dept =d;

}

String getDept() {

return dept;

}

Manager getManager() {

return mgr;

}

Employee[] getEmployee() {

return ely;

}

public String toString() {

return dept + " manager";

}

}

class Test {

public static void main(String[] args) {

Employee[] e1 = {new Employee("Aaron", 50),

new Employee("Betty", 60)};

Manager m1 = new Manager(e1, "Accounting");

Employee[] e2 = {new Employee("Cathy", 70),

new Employee("Dan", 80),

new Employee("Eliz", 90)};

Manager m2 = new Manager(m1, e2, "Production");

System.out.println(m2);

Employee[] emp = m2.getEmployee();

if (emp != null)

for (int k = 0; k < emp.length; k++)

System.out.println(" "+emp[k]+" Salary: $"+ emp[k].getSalary());

Manager m = m2.getManager();

System.out.println(" " + m);

if (m!= null) {

Employee[] emps = m.getEmployee();

if (emps != null)

for (int k = 0; k < emps.length; k++)

System.out.println(" " + emps[k]+" Salary: $"+ emps[k].getSalary());

}

}

}

| C:\ Command Prompt  C:\> java Test  Production manager  Employee Cathy Salary: $70.0  Employee Dan Salary: $80.0  Employee Eliz Salary: $90.0  Accounting manager  Employee Aaron Salary: $50.0  Employee Betty Salary: $60.0 |
| --- |