Centro Universitário UniBH
Ciência da Computação
Práticas de Programação
Professor: Lucas Schmidt

Aula Prática - 8 Padrões Estruturais - Composite

Intenção

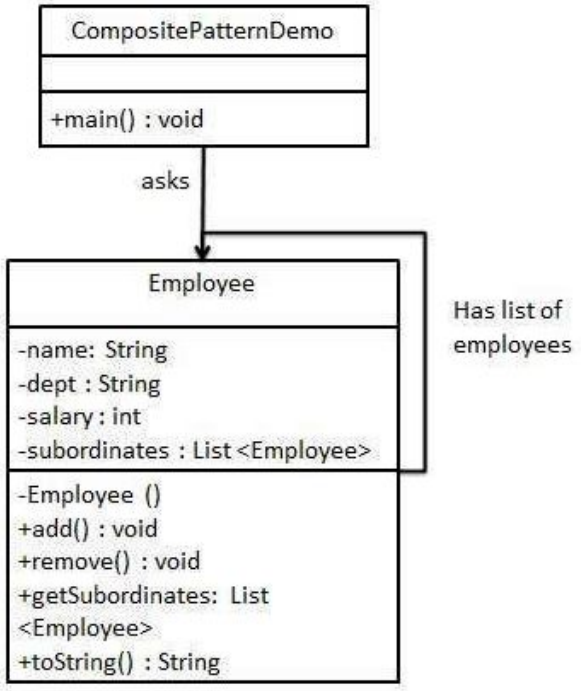
Compor objetos em estruturas de árvore para representar hierarquias todo-parte. Permite que clientes trate objetos individuais e compostos de maneira uniforme.

Usar este padrão quando…

- Quiser representar hierarquias todo-parte;
- Quiser que clientes ignorem a diferença entre objetos simples e objetos compostos.

Vantagens e desvantagens

- Define hierarquias todo-parte:
  - Objetos podem ser compostos de outros objetos e assim por diante.
- Simplifica o cliente:
  - Clientes não se preocupam se estão lidando com compostos ou individuais.
- Facilita a criação de novos membros:
  - Basta estar em conformidade com a interface comum a todos os componentes.
- Pode tornar o projeto muito genérico:
  - Qualquer componente pode ser criado, não há como usar checagem de tipos para restringir.



Passo 1
Criar classe Employee com lista de objetos Employee.

Employee.java

```
import java.util.ArrayList;
import java.util.List;

public class Employee {
   private String name;
   private String dept;
   private int salary;
   private List<Employee> subordinates;

   // constructor
   public Employee(String name,String dept, int sal) {
      this.name = name;
      this.dept = dept;
      this.salary = sal;
      subordinates = new ArrayList<Employee>();
   }

   public void add(Employee e) {
      subordinates.add(e);
   }

   public void remove(Employee e) {
      subordinates.remove(e);
   }

   public List<Employee> getSubordinates(){
     return subordinates;
   }

   public String toString(){
      return ("Employee :[ Name : " + name + ", dept : " + dept + ", salary :" + salary+" ]");
   }
}
```

Passo 2
Use a classe Employee para criar e imprimir a hierarquia de funcionários.

CompositePatternDemo.java

```
public class CompositePatternDemo {
   public static void main(String[] args) {

      Employee CEO = new Employee("John","CEO", 30000);

      Employee headSales = new Employee("Robert","Head Sales", 20000);

      Employee headMarketing = new Employee("Michel","Head Marketing", 20000);

      Employee clerk1 = new Employee("Laura","Marketing", 10000);
      Employee clerk2 = new Employee("Bob","Marketing", 10000);

      Employee salesExecutive1 = new Employee("Richard","Sales", 10000);
      Employee salesExecutive2 = new Employee("Rob","Sales", 10000);

      CEO.add(headSales);
      CEO.add(headMarketing);

      headSales.add(salesExecutive1);
      headSales.add(salesExecutive2);

      headMarketing.add(clerk1);
      headMarketing.add(clerk2);

      //print all employees of the organization
      System.out.println(CEO);

      for (Employee headEmployee : CEO.getSubordinates()) {
         System.out.println(headEmployee);

         for (Employee employee : headEmployee.getSubordinates()) {
            System.out.println(employee);
         }
      }
   }
}
```

Passo 3

Teste sua implementação!