Composite Pattern là một mẫu thiết kế phần mềm thuộc nhóm cấu trúc (Structural Pattern) trong đó một nhóm các đối tượng được xử lý như một đối tượng đơn lẻ. Composite Pattern cho phép tạo ra một cấu trúc cây đối tượng, trong đó các đối tượng đơn lẻ và nhóm đối tượng đều được đối xử như nhau.

Với Composite Pattern, ta có thể tổ chức các đối tượng trong một cấu trúc cây để giải quyết các vấn đề liên quan đến việc quản lý các đối tượng động. Các đối tượng trong cấu trúc cây đều có cùng một giao diện và có thể được truy cập và sử dụng như nhau. Điều này giúp cho việc thêm mới hoặc loại bỏ các đối tượng trong cấu trúc cây trở nên dễ dàng hơn.

Ưu điểm của Composite Pattern:

* Dễ dàng thêm mới hoặc xóa bỏ các đối tượng trong cấu trúc cây.
* Dễ dàng duyệt cấu trúc cây và truy cập các đối tượng.
* Giảm sự phức tạp của việc xử lý các đối tượng động.

Nhược điểm của Composite Pattern:

* Có thể làm tăng độ phức tạp của hệ thống.
* Có thể gây ra hiệu năng kém nếu cấu trúc cây quá lớn.

Ví dụ về Composite Pattern trong mô hình bệnh viện:

Giả sử ta có một hệ thống quản lý bệnh viện, trong đó có các đối tượng như Bác sĩ, Y tá, Bệnh nhân, Phòng khám, Phòng nghỉ, ... Các đối tượng này có thể được sắp xếp theo một cấu trúc cây. Ví dụ, phòng khám có thể chứa các bệnh nhân và các y tá, còn phòng nghỉ có thể chứa các bệnh nhân.

Ta có thể sử dụng Composite Pattern để thiết kế cấu trúc cây cho hệ thống như sau

public interface HospitalUnit {  
 void assign(HospitalUnit hospitalUnit);  
 void remove(HospitalUnit hospitalUnit);  
 List<HospitalUnit> getSubordinates();  
}

public class Hospital implements HospitalUnit {  
 private List<HospitalUnit> hospitalUnits = new ArrayList<HospitalUnit>();  
  
 @Override  
 public void assign(HospitalUnit hospitalUnit) {  
 hospitalUnits.add(hospitalUnit);  
 }  
  
 @Override  
 public void remove(HospitalUnit hospitalUnit) {  
 hospitalUnits.remove(hospitalUnit);  
 }  
  
 @Override  
 public List<HospitalUnit> getSubordinates() {  
 return hospitalUnits;  
 }  
}

public class Department implements HospitalUnit {  
 private String name;  
 private List<HospitalUnit> hospitalUnits = new ArrayList<HospitalUnit>();  
  
 public Department(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public void assign(HospitalUnit hospitalUnit) {  
 hospitalUnits.add(hospitalUnit);  
 }  
  
 @Override  
 public void remove(HospitalUnit hospitalUnit) {  
 hospitalUnits.remove(hospitalUnit);  
 }  
  
 @Override  
 public List<HospitalUnit> getSubordinates() {  
 return hospitalUnits;  
 }  
}

public class Doctor implements HospitalUnit {  
 private String name;  
 private List<HospitalUnit> hospitalUnits = new ArrayList<HospitalUnit>();  
  
 public Doctor(String name) {  
 this.name = name;  
 }  
  
 public void assign(HospitalUnit hospitalUnit) {  
 hospitalUnits.add(hospitalUnit);  
 }  
  
 public void remove(HospitalUnit hospitalUnit) {  
 hospitalUnit.remove(hospitalUnit);  
 }  
  
 public List<HospitalUnit> getSubordinates() {  
 return hospitalUnits;  
 }  
}

Trong ví dụ này, interface HospitalUnit được sử dụng để định nghĩa các hành vi chung cho các đối tượng trong cây. Lớp Hospital, Department và Doctor được thực hiện từ interface này và chứa các phương thức để thêm, xóa và lấy các phần tử con.

Với Composite Pattern, ta có thể dễ dàng tạo ra các cây đối tượng để quản lý các đối tượng trong bệnh viện. Ví dụ, ta có thể tạo một phòng khám chứa các bệnh nhân và các y tá, hoặc tạo một phòng nghỉ chứa các bệnh nhân. Khi ta cần thêm hoặc xóa một bệnh nhân, ta chỉ cần thao tác với phòng khám hoặc phòng nghỉ, mà không cần phải xử lý các bệnh nhân riêng lẻ. Điều này giúp cho việc quản lý các đối tượng trong bệnh viện trở nên dễ dàng hơn.