Filter Pattern là một mẫu thiết kế (design pattern) được sử dụng để lọc các đối tượng trong một tập hợp dữ liệu theo một tập hợp các điều kiện đã được xác định trước đó. Pattern này giúp tách các tiêu chí lọc từ hệ thống của bạn và giúp bạn dễ dàng thêm hoặc xóa bất kỳ tiêu chí nào mà không làm ảnh hưởng đến phần còn lại của hệ thống. Nó còn giúp bạn giảm thiểu số lượng mã lặp lại trong các phương thức lọc và cải thiện khả năng bảo trì của mã của bạn.

Để áp dụng Filter Pattern trong mô hình bệnh viện sử dụng Java, chúng ta có thể có một giao diện lọc (Filter interface) với một phương thức duy nhất là filter(), nó được triển khai bởi các lớp cụ thể lọc khác nhau (Concrete Filter classes) tùy thuộc vào các tiêu chí lọc khác nhau.

Ví dụ, nếu bạn muốn lọc các bệnh nhân theo tuổi, bạn có thể tạo ra một lớp AgeFilter và triển khai phương thức filter() để lọc các bệnh nhân theo độ tuổi. Tương tự, nếu bạn muốn lọc các bệnh nhân theo bệnh lý, bạn có thể tạo ra một lớp DiseaseFilter và triển khai phương thức filter() để lọc các bệnh nhân theo bệnh lý.

Bạn cũng có thể sử dụng một lớp trung gian là FilterChain để thực hiện việc kết hợp các tiêu chí lọc khác nhau. Lớp FilterChain này có thể chứa một danh sách các đối tượng lọc khác nhau và triển khai phương thức filter() để thực hiện việc kết hợp các lọc này lại.

Ví dụ, nếu bạn muốn lọc các bệnh nhân theo cả độ tuổi và bệnh lý, bạn có thể tạo ra một FilterChain với AgeFilter và DiseaseFilter và triển khai phương thức filter() để kết hợp các lọc này lại.

Ưu điểm của Filter Pattern bao gồm:

* Giúp bạn dễ dàng thêm hoặc xóa các tiêu chí lọc mà không ảnh hưởng đến phần còn lại của hệ thống.
* Giảm thiểu sự lặp lại của mã và cải thiện khả năng bảo trì của mã của bạn.
* Cho phép bạn kết hợp các tiêu chí lọc khác nhau để tạo ra những Filter phức tạp hơn, đáp ứng được các yêu cầu phức tạp.
* Là một mẫu thiết kế tiêu chuẩn và được sử dụng rộng rãi trong lập trình phần mềm, do đó nó được nhiều lập trình viên biết đến và sử dụng.

Tuy nhiên, Filter Pattern cũng có một số nhược điểm, bao gồm:

* Tăng độ phức tạp của mã: khi sử dụng Filter Pattern, bạn sẽ phải tạo nhiều lớp cụ thể lọc, điều này có thể dẫn đến tăng độ phức tạp của mã.
* Không thích hợp cho các tập dữ liệu lớn: khi lọc các tập dữ liệu lớn, Filter Pattern có thể gây tốn nhiều tài nguyên hơn do phải lặp lại việc lọc qua từng phần tử.

Ví dụ:

Giả sử bạn đang phát triển một hệ thống quản lý bệnh viện và bạn cần lọc các bệnh nhân theo độ tuổi và bệnh lý. Bạn có thể áp dụng Filter Pattern để giải quyết vấn đề này. Đầu tiên, bạn sẽ tạo một lớp mô tả bệnh nhân và giao diện lọc có tên là PatientFilter với phương thức filter() như sau:

public class Patient {  
 private String name;  
 private int age;  
 private String disease;  
  
 public Patient(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getDisease() {  
 return disease;  
 }  
}

public interface PatientFilter {  
 List<Patient> filter(List<Patient> patients);  
}

Tiếp theo, bạn sẽ tạo hai lớp cụ thể lọc là AgeFilter và DiseaseFilter. Lớp AgeFilter sẽ lọc các bệnh nhân theo độ tuổi và được triển khai như sau:

public class AgeFilter implements PatientFilter {  
 private int minAge;  
 private int maxAge;  
  
 public AgeFilter(int minAge, int maxAge) {  
 this.minAge = minAge;  
 this.maxAge = maxAge;  
 }  
  
 @Override  
 public List<Patient> filter(List<Patient> patients) {  
 List<Patient> filteredPatients = new ArrayList<Patient>();  
  
 for (Patient patient : patients) {  
 if (patient.getAge() >= minAge && patient.getAge() <= maxAge) {  
 filteredPatients.add(patient);  
 }  
 }  
  
 return filteredPatients;  
 }  
}

Lớp DiseaseFilter sẽ lọc các bệnh nhân theo bệnh lý và được triển khai như sau:

public class DiseaseFilter implements PatientFilter{  
 private String disease;  
  
 public DiseaseFilter(String disease) {  
 this.disease = disease;  
 }  
  
 @Override  
 public List<Patient> filter(List<Patient> patients) {  
 List<Patient> filteredPatients = new ArrayList<Patient>();  
  
 for (Patient patient : patients) {  
 if (patient.getDisease().equals(disease)) {  
 filteredPatients.add(patient);  
 }  
 }  
  
 return filteredPatients;  
 }  
}

Cuối cùng, bạn có thể sử dụng FilterChain để kết hợp các bộ lọc với nhau để tạo ra một chuỗi lọc hoàn chỉnh. Ví dụ, bạn có thể tạo một bộ lọc bao gồm lớp AgeFilter và lớp DiseaseFilter như sau:

public class PatientFilterChain {  
 private List<PatientFilter> filters;  
  
 public PatientFilterChain() {  
 this.filters = new ArrayList<PatientFilter>();  
 }  
  
 public void addFilter(PatientFilter filter) {  
 filters.add(filter);  
 }  
  
 public List<Patient> filter(List<Patient> patients) {  
 for (PatientFilter filter : filters) {  
 patients = filter.filter(patients);  
 }  
  
 return patients;  
 }  
}