Flyweight Pattern là một mô hình thiết kế phần mềm được sử dụng để giảm thiểu sự sử dụng bộ nhớ trong các ứng dụng có số lượng đối tượng lớn. Mô hình này tập trung vào việc tái sử dụng các đối tượng có tính chất tương tự để giảm thiểu số lượng các đối tượng được tạo ra.

Các đối tượng được sử dụng trong Flyweight Pattern được chia thành hai loại: intrinsic và extrinsic. Intrinsic là các thuộc tính cố định của đối tượng và không thể thay đổi trong quá trình sử dụng. Extrinsic là các thuộc tính có thể thay đổi của đối tượng và thường được truyền vào đối tượng để sử dụng.

Ví dụ về mô hình bệnh viện:

Giả sử trong mô hình bệnh viện, có rất nhiều bệnh nhân, và mỗi bệnh nhân sẽ có một số thông tin cơ bản như tên, tuổi, địa chỉ, số điện thoại, v.v. Để quản lý thông tin này, có thể tạo một lớp Patient để lưu trữ các thông tin này. Tuy nhiên, nếu có hàng ngàn hoặc hàng triệu bệnh nhân, việc tạo ra các đối tượng Patient riêng biệt cho mỗi bệnh nhân sẽ gây tốn kém về bộ nhớ.

Để giải quyết vấn đề này, có thể sử dụng Flyweight Pattern để giảm thiểu số lượng đối tượng được tạo ra. Thay vì tạo ra một đối tượng Patient mới cho mỗi bệnh nhân, ta có thể sử dụng một đối tượng Patient cố định để lưu trữ các thuộc tính cơ bản của bệnh nhân (intrinsic). Sau đó, các thuộc tính thay đổi (extrinsic) như lịch khám, kết quả xét nghiệm, số tiền thanh toán v.v. sẽ được lưu trữ trong các đối tượng khác và được truyền vào đối tượng Patient khi cần thiết.

Ví dụ :

public class Patient {  
 private String name;  
 private int age;  
 private String address;  
 private String phoneNumber;  
  
 public Patient(String name, int age, String address, String phoneNumber) {  
 this.name = name;  
 this.age = age;  
 this.address = address;  
 this.phoneNumber = phoneNumber;  
 }  
}

public class PatientRecord {  
 private Patient patient;  
 private String appointmentDate;  
 private String medicalTestResult;  
 private double paymentAmount;  
  
 public PatientRecord(Patient patient, String appointmentDate, String medicalTestResult, double paymentAmount) {  
 this.patient = patient;  
 this.appointmentDate = appointmentDate;  
 this.medicalTestResult = medicalTestResult;  
 this.paymentAmount = paymentAmount;  
 }  
}

public class PatientFactory {  
 private static Map<String, Patient> *patientMap* = new HashMap<>();  
  
 public static Patient getPatient(String name, int age, String address, String phoneNumber) {  
 String key = name + "\_" + age + "\_" + address + "\_" + phoneNumber;  
 if (*patientMap*.containsKey(key)) {  
 return *patientMap*.get(key);  
 } else {  
 Patient patient = new Patient(name, age, address, phoneNumber);  
 *patientMap*.put(key, patient);  
 return patient;  
 }  
 }  
}

Trong đó, lớp PatientFactory chịu trách nhiệm quản lý các đối tượng Patient. Mỗi khi có bệnh nhân mới được thêm vào hệ thống, đối tượng Patient sẽ được tạo ra thông qua phương thức getPatient của lớp PatientFactory. Nếu đối tượng Patient đã tồn tại trong bộ nhớ, nó sẽ được trả về, ngược lại, một đối tượng Patient mới sẽ được tạo ra và thêm vào bộ nhớ.

Ưu điểm:

- Giảm thiểu sự sử dụng bộ nhớ bởi vì các đối tượng được chia sẻ và tái sử dụng.

- Tăng tốc độ thực thi chương trình bởi vì các đối tượng đã được tạo sẵn, không cần phải tạo mới khi cần sử dụng.

- Dễ dàng triển khai và tích hợp vào các mô hình phức tạp khác.

Nhược điểm:

- Có thể làm tăng độ phức tạp của mã trong một số trường hợp.

- Không phù hợp với tất cả các tình huống và không thể áp dụng cho tất cả các đối tượng.