**FLYWEIGHT PATTERN**

Chúng ta đã biết Singleton Pattern rất hữu ích trong việc tránh phải khởi tạo nhiều đối tượng không cần thiết gây lãng phí bộ nhớ. Phần này chúng ta sẽ coi 1 pattern khác cũng có tác dụng giảm việc tạo nhiều đối tượng, nguyên tắc của nó là khi làm việc với nhiều đối tượng phức tạp thì ta có thể đưa chúng thành 1 tập các đối tượng chung gọi là flyweights. Một đối tượng phức tạp tốn rất nhiều tài nguyên hệ thống, bằng cách chiết xuất ra những thành phần chung chung của các đồi tượng phức tạp và tin vào runtime configuration để bắt chước các đối tượng phức tạp, chúng ta sẽ tiết kiệm được tài nguyên ???

Chúng ta lấy tất cả các nội dung đặt biệt khỏi đối tượng phức tạp để tạo flyweight object. Khi làm vậy chúng ta làm cho đối tượng hết phức tạp và giảm số lượng cần thiết (có thể chỉ còn một đối tượng).

Flyweight pattern có thể được nói như sau “A flyweight is a shared object that can be used in multiple contexts simultaneously ”

Flyweight pattern có vẻ nghiêng nhiều về hướng thuật toán và tùy trường hợp xử lý do đó ta sẽ coi trực tiếp ví dụ để lấy kinh nghiệm về cách sử dụng flyweight pattern như thế nào. Trong ví dụ này, ta tạo một class Student với các chức năng cần thiết

public class Student

{

**String name;**

**int id;**

**int score;**

**double averageScore;**

**public Student(double a)**

**{**

**averageScore = a;**

**}**

**public void setName(String n)**

**{**

**name = n;**

**}**

**public void setId(int i)**

**{**

**id = i;**

**}**

**public void setScore(int s)**

**{**

**score = s;**

**}**

**public String getName()**

**{**

**return name;**

**}**

**public int getID()**

**{**

**return id;**

**}**

**public int getScore()**

**{**

**return score;**

**}**

**public double getStanding()**

**{**

**return (((double) score) / averageScore - 1.0) \* 100.0;**

**}**

}

Chúng ta chú ý tới phương thức cuối là getStanding trả về vị trí trung bình của sinh viên này.

Bây giờ chúng ta dùng thử flyweight pattern. Vấn đề hiện nay là chúng ta muốn tạo ra nhiều student để xử dụng, thay vì vậy, chúng ta sẽ tạo ra các vùng lưu trữ các tính chất của student (như names, IDs và scores …)

public class TestFlyweight

{

public static void main(String args[])

{

**String names[] = {“Ralph”, “Alice”, “Sam”};**

**int ids[] = {1001, 1002, 1003};**

**int scores[] = {45, 55, 65};**

.

.

.

Để so sánh sinh viên này với sinh viên kia, chúng ta sẽ cần điểm trung bình của các sinh viên.

public class TestFlyweight

{

public static void main(String args[])

{

String names[] = {“Ralph”, “Alice”, “Sam”};

int ids[] = {1001, 1002, 1003};

int scores[] = {45, 55, 65};

**double total = 0;**

**for (int loopIndex = 0; loopIndex < scores.length; loopIndex++){**

**total += scores[loopIndex];**

**}**

**double averageScore = total / scores.length;**

.

.

.

Bây giờ chúng ta tạo một đối tượng có thể hiểu ngầm là flyweight Student object và đưa cho nó số điềm trung bình.

public class TestFlyweight

{

public static void main(String args[])

{

String names[] = {“Ralph”, “Alice”, “Sam”};

int ids[] = {1001, 1002, 1003};

int scores[] = {45, 55, 65};

double total = 0;

for (int loopIndex = 0; loopIndex < scores.length; loopIndex++){

total += scores[loopIndex];

}

double averageScore = total / scores.length;

**Student student = new Student(averageScore);**

.

.

.

Bây giờ chúng ta có thể tính toán vị trí trung bình của nhiều sinh viên khác nhau như sau.

public class TestFlyweight

{

public static void main(String args[])

{

String names[] = {“Ralph”, “Alice”, “Sam”};

int ids[] = {1001, 1002, 1003};

int scores[] = {45, 55, 65};

double total = 0;

for (int loopIndex = 0; loopIndex < scores.length; loopIndex++){

total += scores[loopIndex];

}

double averageScore = total / scores.length;

Student student = new Student(averageScore);

**for (int loopIndex = 0; loopIndex < scores.length; loopIndex++){**

**student.setName(names[loopIndex]);**

**student.setId(ids[loopIndex]);**

**student.setScore(scores[loopIndex]);**

**System.out.println(“Name: “ + student.getName());**

**System.out.println(“Standing: “ +**

**Math.round(student.getStanding()));**

**System.out.println(“”);**

**}**

}

}

Kết quả xuất ra là

Name: Ralph

Standing: -18

Name: Alice

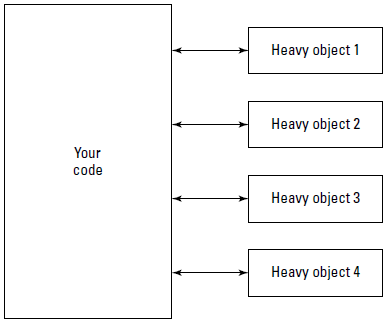
Standing: 0

Name: Sam

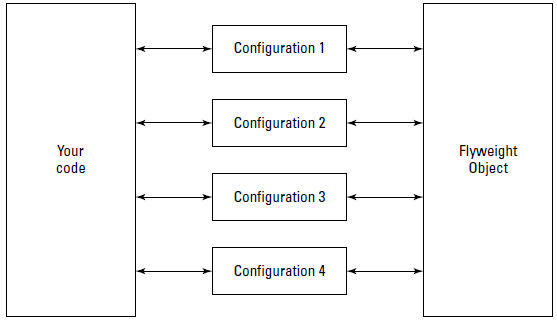
Standing: 18

Nhận xét : thay vì tạo 3 đối tượng (với tư duy thuần OOP), chúng ta có thể hạn chế và chỉ cần tạo 1 đối tượng cũng có thể giải quyết được bài toán. Cũng giống như singleton pattern, flyweight pattern kiểm soát các đối tượng và giảm thiểu số lượng phát sinh không cần thiết mà vẫn giải quyết tốt bài toán. (Chú ý là trong trường hợp này ta có thể không thể xài flyweight pattern nếu bài toán phức tạp hơn)

Ta coi lại mô tả flyweight pattern



Hình trên là nguyên gốc code sử dụng tư tương thuần OOP



Hình trên sau khi đã sử dụng Flyweight Object tránh phải tạo quá nhiều đối tượng mà vẫn giải quyết được bài toán