**SINGLETON PATTERN**

Trong phần này chúng ta coi 1 trong số các pattern được sử dụng rất nhiều và quan trọng là singleton pattern. Nội dung của singleton pattern là tạo ra 1 calss có thể xây dựng 1 đối tượng duy nhất để tránh việc sản sinh ra nhiều đối tượng khác nhau trong chương trình. Nhưng tại sao lại cần phải như thế, ta coi một ví dụ sau đây

class ResourceManager

{

private:

int x;

public:

ResourceManager(){x =0;....}

void SetX(int \_x){ x = \_x;}

void GetX(){ return x;}

}

Trên đây là class ResourceManager, chúng ta hãy coi cách sử dụng resource trên như thế nào

class ClientObject1 {

public:

void DoSomething()

{

**ResourceManager rm;**

printf(“x = %d”,rm.GetX());

x = 100;

} }

class ClientObject2

{

public:

void DoSomething()

{

**ResourceManager rm;**

printf(“x = %d”,rm.GetX());

x = 500;

} }

Ví dụ trên cho ta thấy thực tế là ta đã gọi resource 2 lần và ứng với 2 lần gọi đó là 2 đối tượng rm khác nhau được tạo ra, 2 đối tượng này sẽ giữ giá trị x khác nhau là 100 và 500. Rõ ràng ta không muốn có được kết quả như trên vì ta muốn 2 Object chỉ truy cập vào cùng 1 đối tượng.

Để giải quyết vấn đề này, ta sẽ áp dụng Singleton Pattern để đảm bảo dù được gọi vào bao nhiều lần thì nó vẫn đảm bảo cho ta chỉ duy nhất một đối tượng được phát sinh.

Chúng ta sẽ xây dựng 1 class Database áp dụng singleton Pattern để đảm bảo chỉ duy nhất 1 đối tượng Database được phát sinh. Chúng ta xem qua một chút tại sao lại có thể phát sinh đối tượng cùng loại nhiều lần. Nguyên nhân chủ yếu nằm ở constructor của class đó, có nghĩa là mỗi lần gọi constructor cũng đồng nghĩa với việc tạo 1 đối tượng mới. Như vậy điều ta cần làm là không cho phép gọi constructor. Nhưng constructor là cái không thể thiếu đối với 1 class. Vậy điều duy nhất có thể làm với constructor ở đây là đưa nó thành private. Nghe có vẻ vô lý nhưng thực tế là có chuyện đó như các đối tượng vẽ Graphic hay Graphic2D đều bảo vệ contructor để tránh phát sinh nhiều đối tượng. Ok bây giờ chúng ta bắt đầu xây dựng class Database

public class Database

{

private int record;

private String name;  
**private Database(String n)**

{

name = n;

record = 0;

}

Tiếp theo ta thấy không có constructor do đó ta phải tạo hàm để lấy đối tượng về (tạo con trỏ đến đối tượng) để ta sử dụng

public class Database

{

private int record;

private String name;

private Database(String n)

{

name = n;

record = 0;

}

**public static Database getInstance(String n)**

**{**

**}**

.

.

.

}

Hàm getInstance có tác dụng như hàm khởi tạo với 1 tham số string n và trả về đối tượng Database duy nhất, đối tượng duy nhất này cũng là đối tượng ta để sẵn trong Database. Ta coi đoạn code

public class Database

{

**private static Database singleObject;**

private int record;

private String name;

private Database(String n)

{  
name = n;

record = 0;

}

public static Database getInstance(String n)

{

**if (singleObject == null){**

**singleObject = new Database(n);**

**}**

**return singleObject;**

}

.

.

.

}

Như ta thấy đối tượng duy nhất ở đây là **singleObject**. Như vậy thì dù có gọi nhiều lần thì nó vẫn sẽ trả về là đối tượng singleObject. Chúng ta hãy thử test class trên

public class TestSingleton

{

public static void main(String args[])

{

Database database;

database = Database.getInstance(“products”);

System.out.println(“This is the “ +

database.getName() + “ database.”);

**database = Database.getInstance(“employees”);**

**System.out.println(“This is the “ +**

**database.getName() + “ database.”);**

}

}

Kết quả xuất ra là như nhau.

This is the products database.

This is the products database.