**FACTORY PATTERN**

Trong phần này chúng ta sẽ coi qua cách sử dụng factory pattern trong một ví dụ.

Cấu trúc chương trình cần như sau, chúng ta có đối tượng Factory trong đó có hàm createConnection sẽ tạo kết nối tới CSDL. Vấn đề sẽ đơn giản nếu ta chỉ cần tạo một kết nối tới CSDL SQL và ta có thể code ngay vào class Factory, tuy nhiên vấn đề phát sinh ở đây là ta muốn tạo ra kết nối tới nhiều CSDL khác nhau như Oracle, SQL, MySQL … Như vậy, một điều phải làm là ta phải kiểm tra coi người dùng mún tạo kết nối tới loại CSDL nào (kiểm tra bằng if hoặc switch) sau đó xây dựng loại kết nối phù hợp, như vậy nếu cho khả năng kết nối tới 3 loại CSDL khác nhau thì ta phải code ra 3 chức năng. Vấn đề là nếu ta đặt tất cả trong hàm createConnection thì sẽ làm cho hàm quá rắc rối, đó là chưa kể đến việc tạo hoặc nâng cấp thêm các loại CSDL khác. Do đó người ta đưa ra giải pháp Factory Pattern, nguyên lý của nó như cái tên của nó, bản thân đối tượng được xây dựng theo Factory Pattern sẽ phát sinh ra các đối tượng khác khi gọi hàm thích hợp.

Quay trở lại với ví dụ, đối tượng Factory sẽ phát sinh ra 1 đối tượng mới khi được gọi hàm createConnection, đối tượng khác này ta sẽ gọi là Connection.

Một chú ý khác là ta sẽ có thể xây dựng 2 kiểu factory pattern khác nhau. Do từ lúc tạo đối tượng đến phát sinh đối tượng mới ta phải thông qua giai đoạn chọn loại CSDL nên ta có 2 cách thực thi factory pattern.

* Thứ 1 là tạo đối tượng Factory phù hợp với loại CSDL cần rồi gọi hàm createConnection để tạo đối tượng Connection phù hợp
* Thư 2 là tạo đối tượng Factory bình thường rồi gọi hàm createConnection với loại CSDL phù hợp (Khuyến khích dùng cách 2 vì nó mềm mại hơn cách 1)

Ta coi qua cách tạo factory pattern theo hướng 1. Chúng ta coi qua class Factory

public class FirstFactory

{

protected String type;

public FirstFactory(String t)

{

type = t;

}

**public Connection createConnection()**

**{**

**if (type.equals(“Oracle”)){**

**return new OracleConnection();**

**}**

**else if (type.equals(“SQL Server”)){**

**return new SqlServerConnection();**

**}**

**else {**

**return new MySqlConnection();**

**}**

**}**

}

Như ta thấy ở trên việc tạo đối tượng (construction) sẽ đi kèm với biến string quyết định loại kết nối CSDL. Tùy theo loại kết nối CSDL mà tao sẽ cho hàm createConnection tạo ra đối tượng Connection thích hợp. Ta coi thử cách sử dụng các class này

FirstFactory factory;

factory = new FirstFactory(“Oracle”);

**Connection connection = factory.createConnection();**

**connection.setParams(“username”, “Steve”);**

**connection.setParams(“password”, “Open the door!!!)”;**

**connection.initialize();**

**connection.open();**

.

.

.

Ví dụ trên tạo một đối tượng factory với kết nối CSDL kiểu Oracle.

Vấn đề ở đây là ta sẽ xây dựng class Connection như thế nào, ta dễ dàng nhận thấy Connection có khả năng chuyển đổi thành các kiểu Connection khác nhau, nói cách khác có nhiểu kiểu đối tượng khác nhau được tạo ra thông qua class Connection. Ta sẽ xây dựng điều như trên bằng cách xây dựng 1 class ảo Connection và có nhiều class con thừa kế class ảo này. Đầu tiên là class Connection

public abstract class Connection

{

public Connection()

{

}

public String description()

{

return “Generic”;

}

}

Sau đó là các class thừa kế

public class OracleConnection extends Connection

{

public OracleConnection()

{

}

public String description()

{

return “Oracle”;

}

}

public class SqlServerConnection extends Connection

{

public SqlServerConnection()

{

}

public String description()

{

return “SQL Server”;

}

}

public class MySqlConnection extends Connection

{

public MySqlConnection()

{

}

public String description()

{

return “MySQL”;

}

}

Bây giờ chúng ta chuyển sang coi thử cách 2. Để phân biệt một chút so với cách 1 ta đổi tên class Factory thành ConnectionFactory. Ngoài ra ta xây dựng ConnectionFactory là 1 class ảo để cho thấy tính mềm dẻo và khả năng mở rộng.

public abstract class ConnectionFactory

{

public ConnectionFactory()

{

}

**protected abstract Connection createConnection(String type);**

}

Do là class ảo nên ta cần 1 class kế thừa nó.

public class SecureFactory extends ConnectionFactory

{

public Connection createConnection(String type)

{

**if (type.equals(“Oracle”)){**

**return new SecureOracleConnection();**

**}**

**else if (type.equals(“SQL Server”)){**

**return new SecureSqlServerConnection();**

**}**

**else {**

**return new SecureMySqlConnection();**

**}**

}

}

Phần còn lại là class Connection và các class kế thừa giống như ở trên đã nói.

public class SecureOracleConnection extends Connection

{

public SecureOracleConnection()

{

}

**public String description()**

**{**

**return “Oracle secure”;**

**}**

}

public class SecureSqlServerConnection extends Connection

{

public SecureSqlServerConnection()

{

}

**public String description()**

**{**

**return “SQL Server secure”;**

**}**

}

public class SecureMySqlConnection extends Connection

{

public SecureMySqlConnection()

{

}

**public String description()**

**{**

**return “MySQL secure”;**

**}**

}

Chúng ta sẽ test thử class vừa tạo

public class TestFactory

{

public static void main(String args[])

{

SecureFactory factory;

factory = new SecureFactory();

**Connection connection = factory.createConnection(“Oracle”);**

**System.out.println(“You’re connecting with “ +**

**connection.description());**

}

}