6.15 Mẫu thiết kế PROTOTYPE

6.15.1 Mở đầu

Trong lập trình hướng đối tượng, chúng ta cần các đối tượng để làm việc với chúng; các đối tượng tương tác với nhau để hoàn thành công việc. Nhưng đôi khi, tạo ra một đối tượng nặng có thể phải trả giá đắt, và nếu ứng dụng của bạn cần quá nhiều các đối tượng (chứa hầu hết các tính chất tương tự), nó có thể tạo ra một số vấn đề về hiệu năng.

Giả sử chúng ta xem xét kịch bản sau, ở đó ứng dụng yêu cầu kiểm soát quyền truy cập. Các đặc tính này của ứng dụng có thể được sử dụng bởi các người sử dụng tuân thủ theo quyền truy cập được cấp cho họ. Chẳng hạn, một số người sử dụng được quyền truy cập đến báo cáo được sinh bởi ứng dụng, trong khi một số khác thì không. Một số trong họ ngay cả có thể sửa báo cáo, trong khi một số chỉ có thể đọc nó. Một số người sử dụng cũng có quyền hành chính như bổ sung hoặc loại bỏ các người sử dụng khác.

Mỗi đối tượng người sử dụng sẽ có đối tượng kiểm soát quyền truy cập, mà được sử dụng để cung cấp hoặc hạn chế quyền truy cập của ứng dụng. Đối tượng kiểm soát quyền truy cập này là đối tượng đồ sộ, nặng và việc tạo ra nó rất tốn kém, vì nó yêu cầu dữ liệu được đẩy vào từ tài nguyên bên ngoài nào đó, như cơ sở dữ liệu hoặc một số file về sở hữu.

Chúng ta cũng không thể chia sẻ đối tượng quyền truy cập với người sử dụng cùng cấp độ, vì các quyền có thể thay đổi trong thời gian chạy bởi người quản trị và người sử dụng ở cùng cấp độ có thể có các quyền truy cập khác nhau. Một đối tượng người sử dụng có một đối tượng quyền truy cập.

Chúng ta có thể sử dụng mẫu thiết kế *Prototype* để giải quyết bài toán này bằng cách tạo các đối tượng kiểm soát truy cập trên mọi mức độ một lần và sau đó cung cấp bản sao của đối tượng này cho người sử dụng khi được yêu cầu. Trong trường hợp này, dữ liệu đẩy vào từ tài nguyên bên ngoài chỉ xảy ra một lần. Lần sau, đối tượng kiểm soát quyền truy cập được tạo bằng cách copy một đối tượng đã có. Đối tượng kiểm soát quyền truy cập này là không được tạo từ đầu mỗi lần yêu cầu được gửi đi, cách tiếp cận này sẽ giảm thời gian tạo đối tượng.

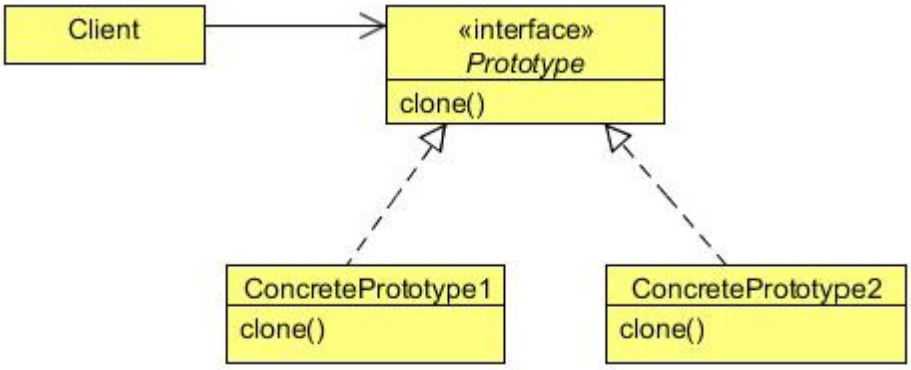
Trước khi đưa ra giải pháp, chúng ta sẽ tìm hiểu thêm về mẫu thiết kế Prototype.

6.15.2 Mẫu thiết kế Prototype là gì

Mẫu thiết kế Prototype được sử dụng để đặc tả mọi kiểu đối tượng được tạo sử dụng một khởi tạo nguyên mẫu và tạo đối tượng mới bằng cách copy bản mẫu này.

Khái niệm là copy một đối tượng đã có thay vì tạo mới một đối tượng từ đầu, đôi khi có thể bao gồm các thao tác đắt giá. Các đối tượng đã có đóng vai trò như nguyên mẫu và chứa trạng thái của đối tượng. Đối tượng mới được sao chép có thể thay đổi một số tính chất chỉ nếu nó được yêu cầu. Cách tiếp cận này tiết kiệm tài nguyên và thời gian đáng kể, đặc biệt khi việc tạo đối tượng là quá trình phức tạp.

Trong Java, có một số cách copy một đối tượng để tạo cái mới. Một cách đạt được điều này là sử dụng giao diện Cloneable. Java cung cấp phương thức clone, mà đối tượng kế thừa từ lớp Object. Bạn cần cài đặt giao diện Cloneable và viết đè phương thức clone này theo yêu cầu của bạn.



Prototype

- Khai báo một giao diện để sao chép chính nó

ConcretePrototype

- Cài đặt thao tác *clone* để sao chép chính nó

Client

- Tạo một đối tượng mới bằng cách yêu cầu một *prototype clone* chính nó

Prototype cho phép bạn kết hợp một lớp *product* cụ thể mới vào một hệ thống đơn giản bằng cách đăng ký một khởi tạo nguyên mẫu với *client*.

6.15.3 Giải pháp cho bài toán

Trong lời giải này, chúng ta sẽ sử dụng phương thức *clone* để giải quyết bài toán trên

```
package com.javacodegeeks.patterns.prototypepattern;

public interface Prototype extends Cloneable {

        public AccessControl clone() throws CloneNotSupportedException;

}
```

Giao diện trên mở rộng giao diện *Cloneable* và chứa phương thức clone. Giao diện này được cài đặt bởi các lớp mà muốn tạo đối tượng nguyên mẫu *prototype*.

```
package com.javacodegeeks.patterns.prototypepattern;

public class AccessControl implements Prototype{

        private final String controlLevel;
        private String access;

        public AccessControl(String controlLevel,String access){
                this.controlLevel = controlLevel;
                this.access = access;
        }
```

```
        @Override
        public AccessControl clone(){
                try {
                        return (AccessControl) super.clone();
                } catch (CloneNotSupportedException e) {
                        e.printStackTrace();
                }
                return null;
        }

        public String getControlLevel(){
                return controlLevel;
        }

        public String getAccess() {
                return access;
        }

        public void setAccess(String access) {
                this.access = access;
        }

}
```

Lớp *AccessControl* cài đặt giao diện *Prototype* và ghi đè phương thức *clone*. Phương thức này gọi phương thức clone của lớp cha và trả về đối tượng sau khi ép xuống về kiểu *AccessControl*. Phương thức *clone* đưa ra báo lỗi *CloneNotSupportedException* bên trong bản thân phương thức.

Lớp này cũng chứa hai tính chất; *controlLevel* được sử dụng để chuyên biệt mức kiểm soát mà đối tượng chứa. Mức này phụ thuộc vào kiểu người sử dụng mà dùng nó, chẳng hạn, USER, ADMIN, MANAGER.

Ma trận kiểm soát quyền truy cập

	Program1	...	SegmentA	SegmentB
Process1	Read Execute		Read Execute	
Process2				Read

Tính chất khác là *access*. Nó chứa quyền truy cập cho người sử dụng. Lưu ý rằng, ở đây để đơn giản, chúng ta sử dụng *access* như thuộc tính kiểu *String*. Nó có thể là kiểu *Map* mà có thể chứa cặp khóa-giá trị của danh sách dài quyền truy cập được trao cho người sử dụng.

```
package com.javacodegeeks.patterns.prototypepattern;

public class User {

        private String userName;
        private String level;
        private AccessControl accessControl;

        public User(String userName,String level, AccessControl accessControl){
                this.userName = userName;
                this.level = level;
                this.accessControl = accessControl;
        }

        public String getUserName() {
                return userName;
        }
        public void setUserName(String userName) {
                this.userName = userName;
        }
        public String getLevel() {
                return level;
        }
        public void setLevel(String level) {
                this.level = level;
        }
```

```
        public AccessControl getAccessControl() {
                return accessControl;
        }
        public void setAccessControl(AccessControl accessControl) {
                this.accessControl = accessControl;
        }

        @Override
        public String toString(){
                return "Name: "+userName+", Level: "+level+", Access Control Level:"+ ↩
                    accessControl.getControlLevel()+", Access: "+accessControl.getAccess();
        }

}
```

Lớp *User* có *username*, *level* và một tham chiếu đến *AccessControl* được gán cho nó.

Chúng ta đã sử dụng lớp *AccessControlProvider* mà tạo và lưu giữ các đối tượng *AccessControl* từ trước. Và khi có yêu cầu đến đối *tượng AccessControl*, nó trả về đối tượng mới được tạo bởi *copy* các nguyên mẫu đã lưu.

```
package com.javacodegeeks.patterns.prototypepattern;

import java.util.HashMap;
import java.util.Map;

public class AccessControlProvider {

        private static Map<String, AccessControl>map = new HashMap<String, AccessControl>() ↩
            ;

        static{

                System.out.println("Fetching data from external resources and creating  ↩
                    access control objects...");
                map.put("USER", new AccessControl("USER","DO_WORK"));
                map.put("ADMIN", new AccessControl("ADMIN","ADD/REMOVE USERS"));
                map.put("MANAGER", new AccessControl("MANAGER","GENERATE/READ REPORTS"));
                map.put("VP", new AccessControl("VP","MODIFY REPORTS"));
        }

        public static AccessControl getAccessControlObject(String controlLevel){
                AccessControl ac = null;
                ac = map.get(controlLevel);
                if(ac!=null){
                        return ac.clone();
                }
                return null;
        }
}
```

Phương thức *getAccessControlObjec*t đưa ra một đối tượng nguyên mẫu được lưu giữ tùy theo *controlLevel* truyền cho nó, từ map đó và trả về đối tương được nhân bản tạo mới cho code của client.

Bây giờ chúng ta kiểm tra code trên.

```
package com.javacodegeeks.patterns.prototypepattern;

public class TestPrototypePattern {

        public static void main(String[] args) {
                AccessControl userAccessControl = AccessControlProvider. ↩
                    getAccessControlObject("USER");
                User user = new User("User A", "USER Level", userAccessControl);
```

```
                System.out.println("**********************************");
                System.out.println(user);

                userAccessControl = AccessControlProvider.getAccessControlObject("USER");
                user = new User("User B", "USER Level", userAccessControl);
                System.out.println("Changing access control of: "+user.getUserName());
                user.getAccessControl().setAccess("READ REPORTS");
                System.out.println(user);

                System.out.println("**********************************");

                AccessControl managerAccessControl = AccessControlProvider. ↩
                    getAccessControlObject("MANAGER");
                user = new User("User C", "MANAGER Level", managerAccessControl);
                System.out.println(user);
        }
}
```

Nó cho kết quả đầu ra như sau:

```
Fetching data from external resources and creating access control objects...
**********************************
Name: User A, Level: USER Level, Access Control Level:USER, Access: DO_WORK
Changing access of: User B
Name: User B, Level: USER Level, Access Control Level:USER, Access: READ REPORTS
**********************************
Name: User C, Level: MANAGER Level, Access Control Level:MANAGER, Access: GENERATE/READ  ↩
    REPORTS
```

Trong code trên, chúng ta đã tạo đối tượng *AccessControl* tại mức USER và gán nó cho *User* A. Sau đó lại đối tượng *AccessControl* khác User B, nhưng lần này chúng ta thay đổi quyền truy cập của User B. Cuối cùng, quyền truy cập mức MANAGER cho User C.

Đối tượng *getAccessControlObject* được sử dụng để lấy bản sao mới của đối tượng *AccessControl* và điều này được nhìn thấy rõ khi chúng ta thay đổi quyền truy cập của User B, quyền truy cập của User A không thay đổi. Điều này được khẳng định rằng phương thức *clone* làm việc đúng, như nó trả về bản copy mới của đối tượng chứ không phải tham chiếu mà trỏ đến cùng một đối tượng.

6.15.4 Khi nào dùng mẫu thiết kế Prototype

Sử dụng mẫu thiết kế *Prototype* khi một hệ thống cần là độc lập với việc các sản phẩm của nó được tạo, kết hợp và thể hiện như thế nào và

- Khi các lớp để khởi tạo được đặc tả trong thời gian thực, chẳng hạn, bằng tải động hoặc
- Để tránh việc xây dựng một phân cấp lớp các factory mà chạy song song với phân cấp lớp của sản phẩm.
- Khi các khởi tạo của một lớp có thể có một trong chỉ một số ít kết hợp khác nhau của trạng thái. Nó có thể là thuận tiện hơn để cài đặt một số tương ứng các nguyên mẫu và nhân bản clone chúng hơn là khởi tạo lớp bằng tay, mỗi lần với một trạng thái phù hợp.