TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

# Tài liệu đặc tả yêu cầu phần mềm

(Software Requirement Specification – SRS)

# COURSE REGISTRATION

Môn: Phát triển phần mềm theo chuẩn kĩ năng ITSS

**Nhóm số 04**

**Danh sách sinh viên**


| | |
|---|---|
| Phạm Đức Hiển | 20141623 |
| Nguyễn Văn Hãnh | 20141418 |
| Đỗ Việt Hưng | 20131936 |


*Hà Nội, ngày 03 tháng 12 năm 2017*

# Báo cáo Design pattern

## Nhóm isd.vn.20171-04

State Pattern ( Phạm Đức Hiển)

Memento Pattern ( Nguyễn Văn Hãnh)

Proxy Pattern ( Đỗ Việt Hưng)

# I. State pattern:

## 1. Giới thiệu:

- **Trong mẫu State pattern, hành vi của một lớp thay đổi dựa theo trạng thái của chính nó.**
- **State pattern là một kiểu behavior pattern.**
- **Trong State pattern, chúng ta tạo các lớp đại diện cho nhiều trạng thái khác nhau của một đối tượng. Và các đối tượng sẽ có hành vi thay đổi dựa trên trạng thái của chính nó.**

## 2. Ví dụ bài toán:

- **Một game gồm lần lượt các trạng thái: Đang load (LoadingState), đang chơi (PlayingState), đang dừng (PausingState).**


- **Việc quản lí các trạng thái của Game sẽ phải dùng một biến state để lưu trạng thái của nó. Và thông thường sẽ phải dùng rất nhiều lệnh if, else hoặc switch case.**
- **Nếu số lượng các trạng thái, và hành vi là lớn thì việc quản lí từng hành vi lại có chi phí rất lớn.**

**=> Giải pháp : Sử dụng State pattern, tuỳ theo trạng thái hiện tại của Game mà nó sẽ có hành vi chuyển đổi trạng thái tương ứng.**

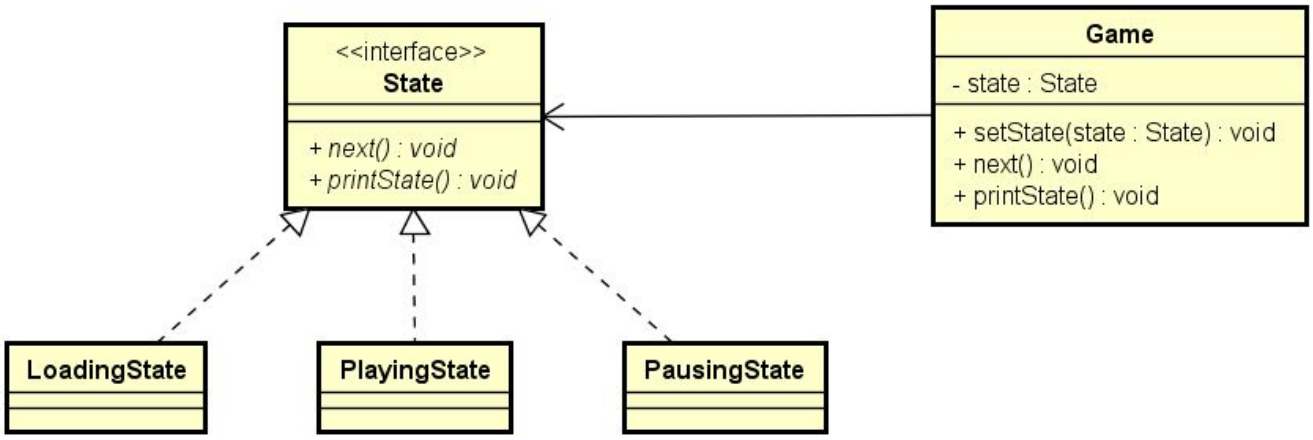
## 3. Áp dụng State pattern:

- **Tạo một interface State chức 2 phương thức :**

- **public void next(Game game) để chuyển trạng thái tiếp theo.**
- **printState() để in trạng thái hiện tại.**

- **Tạo ba lớp LoangdingState, PlayingState và PausingState đều implements interface State.**
- **Tạo một lớp Game chứa thuộc tính private State state.**
- **Ta sẽ thực hiện xử lí hành vi của lớp Game dựa theo thuộc tính state.**

- **Biểu đồ UML:**




## a. Class Game:

```
1
2   package statedemo;
3   /**
4    *
5    * @author HPD
6    */
7   public class Game {
8       private State state;
9
10      public void setState(State state) {
11          this.state = state;
12      }
13
14      public void next() {
15          this.state.next(this);
16      }
17      public void printState() {
18          this.state.printState(this);
19      }
20  }
```

## B. Interface State:

```
1
2   package statedemo;
3
4   /**
5    *
6    * @author HPD
7    */
    public interface State {
        public void next(Game game);
        public void printState(Game game);
11  }
12
```

### c. Class LoadingState implements State:

```
1
2   package statedemo;
3
4   /**
5    *
6    * @author HPD
7    */
8   public class LoadingState implements State {
9
10      @Override
        public void next(Game game) {
12          game.setState(new PlayingState());
13      }
14
15      @Override
        public void printState(Game game) {
17          System.out.println("Game is Loading...");
18      }
19  }
20
```

## D. Class PlayingState implements State:

```
1
2   package statedemo;
3
4   /**
5    *
6    * @author HPD
7    */
8   public class PlayingState implements State {
9
10      @Override
11      public void next(Game game) {
12          game.setState(new PausingState());
13      }
14      @Override
15      public void printState(Game game) {
16          System.out.println("Game is Playing...");
17      }
18  }
19
```

## E. Class PausingState implements State:

```java
package statedemo;

/**
 *
 * @author HPD
 */
public class PausingState implements State {

    @Override
    public void next(Game game) {
        game.setState(new PlayingState());
    }
    @Override
    public void printState(Game game) {
        System.out.println("Game is Pausing...");
    }
}
```

## F. Demo:

```java
package statedemo;
public class StateDemo {
    public static void main(String[] args) {
        Game game = new Game();
        LoadingState loadState = new LoadingState();
        game.setState(loadState);

        game.printState(); // set loading

        game.next(); // change to playing
        game.printState();  // state playing

        game.next(); // change to pausing
        game.printState();  // state pausing
    }
}
```

```
run:
Game is Loading...
Game is Playing...
Game is Pausing...
```

## 4. Kết luận:

**State pattern rất phù hợp với các bài toán yêu cầu hành vi của đối tượng thay đổi tuỳ theo trạng thái của nó. Đặc biệt State pattern tỏ ra cực kì hữu ích trong các bài toán mà đối tượng có nhiều trạng thái phức tạp và tương ứng với các trạng thái đó lại thực hiện các hành vi khác nhau.**

# II. Proxy Design Pattern

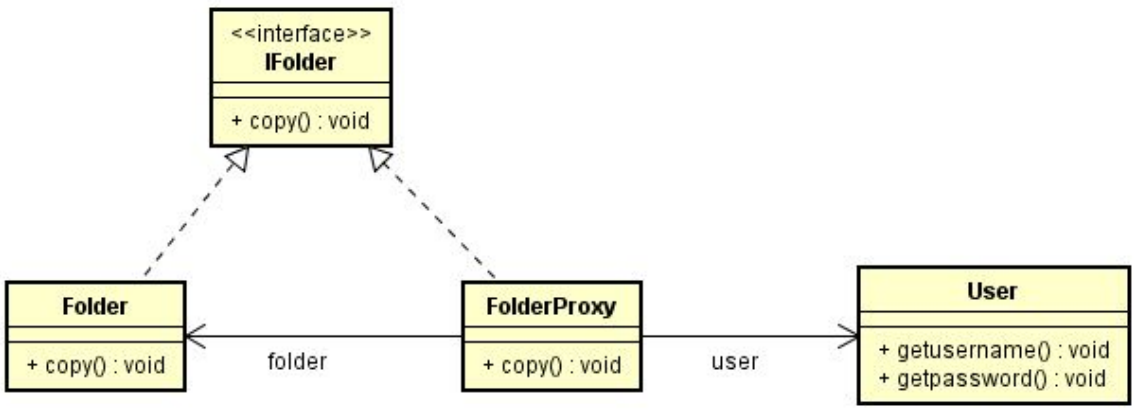
## 1. Giới thiệu

design pattern- proxy đại diện cho chức năng của 1 lớp khác để thực hiện chức năng của nó ở bên ngoài

## 2. khi nào cần dùng design pattern proxy

- Khi muốn bảo vệ quyền truy xuất vào các chức năng (phương thức) của thực thể
- Bổ sung trước khi thực hiện phương thức của thực thể.
- Tạo đối tượng với chức năng được nâng cao theo yêu cầu.

## 3. UML class Diagram




## 4. Ví Dụ code minh họa

### a giới thiệu

Trong ví dụ có một thư mục có thể thực hiện các thao tác sao chép. chúng ta có `IFolder` giao diện và `Folder` lớp cung cấp `coppy()` phương thức, và đây là lớp và giao diện hiện tại mà chúng

ta không thể thay đổi. Chúng tôi muốn chỉ ra thêm rằng chỉ có người dùng có quyền truy cập mới có thể truy cập vào nó và thực hiện các thao tác như cắt hoặc sao chép các tệp và thư mục con.

# b.code

## -IFolder.java

```
package proxyDP;

/**
 *
 * @author Viet Hưng
 */
public interface IFolder {
    public void coppy();
}
```

## -folder.java

```
        package proxyDP;

/**
 *chúng ta không thể thay đổi nhưng chúng ta có thể cấp quyền trên nó
 * @author Viet Hung
 */
public class Folder implements IFolder {
     public void coppy()
 {
       //truy cập vào thư mục và thực hiện sao chép
  System.out.println("thực hiện coppy trên thư mục");
 }

}
```

## -Folderproxy.java

```
package proxyDP;

/**
```

*cung cấp  phép cho lớp `Folder`. Nó kiểm tra `username` và `password` và nếu kết hợp, sau đó chỉ có nó cho phép truy cập vào thư mục.

```
 * @author Viet Hung
 */
public class FolderProxy implements IFolder {
    Folder folder;
 User user;

 public FolderProxy(User user) {
  this.user = user;
 }

    /**
     *
     */
    @Override
 public void coppy() {

  if(user.getUserName().equalsIgnoreCase("admin") &&
  user.getPassword().equalsIgnoreCase("admin"))
  {
   folder=new Folder();
   folder.coppy();
  }
  else
  {
   System.out.println("bạn không thể làm gì");
  }
      }

}
```

## -User.java

```
package proxyDP;

/**
 *
 * @author Viet Hưng
 */
public class User {
    String userName;
```

```
 String password;

 public User(String userName, String password) {
  this.userName = userName;
  this.password = password;
 }

 public String getUserName() {
  return userName;
 }
 public String getPassword() {
  return password;
 }


}
```

## -main.java

```
package proxyDP;

/**
 *
 * @author Viet Hung
 */
public class main {
    public static void main(String[] args) {

  // Khi bạn click vào thư mục, hệ thống sẽ hỏi userName và mật khẩu.
  //và hệ thống sẽ tạo đối tượng người dùng này

  //nếu cung cấp đúng mật khẩu
  User user=new User("admin","admin");
  FolderProxy folderProxy=new FolderProxy(user);
  System.out.println("Khi userName và password chính xác:");
  folderProxy.coppy();
  System.out.println("**************************************");
  // if we give wrong userName and Password
  User userWrong=new User("abc","abc");
  FolderProxy folderProxyWrong=new FolderProxy(userWrong);
  System.out.println(" Khi userName và password không chính xác:");
```

```
  folderProxyWrong.coppy();
 }

}
```

# III. Proxy Design Pattern

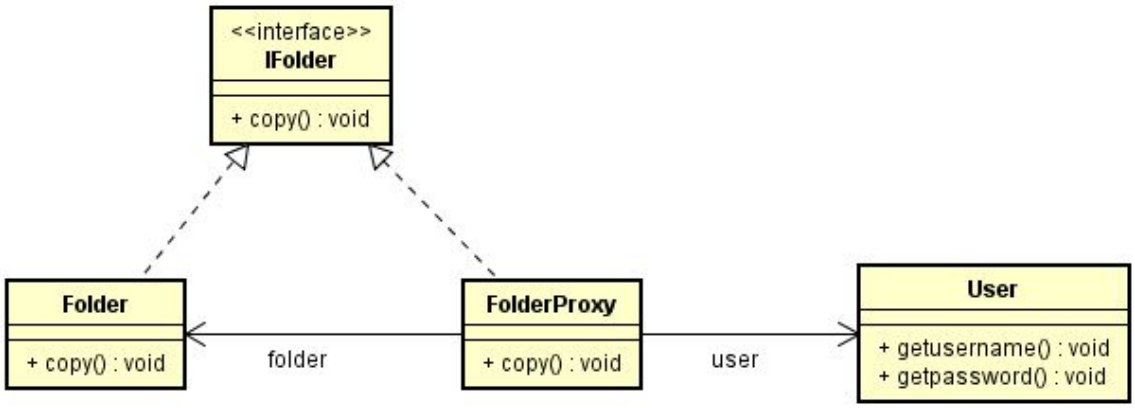
## 1. Giới thiệu

design pattern- proxy đại diện cho chức năng của 1 lớp khác để thực hiện chức năng của nó ở bên ngoài

## 2. khi nào cần dùng design pattern proxy

- Khi muốn bảo vệ quyền truy xuất vào các chức năng (phương thức) của thực thể
- Bổ sung trước khi thực hiện phương thức của thực thể.
- Tạo đối tượng với chức năng được nâng cao theo yêu cầu.

## 3. UML class Diagram




## 4. Ví Dụ code minh họa

### a giới thiệu

Trong ví dụ có một thư mục có thể thực hiện các thao tác sao chép. chúng ta có `IFolder` giao diện và `Folder` lớp cung cấp `coppy()` phương thức, và đây là lớp và giao diện hiện tại mà chúng ta không thể thay đổi. Chúng tôi muốn chỉ ra thêm rằng chỉ có người dùng có quyền truy cập mới có thể truy cập vào nó và thực hiện các thao tác như cắt hoặc sao chép các tệp và thư mục con.

# b.code

## -IFolder.java

```
package proxyDP;

/**
 *
 * @author Viet Hưng
 */
public interface IFolder {
    public void coppy();
}
```

## -folder.java

```
        package proxyDP;

/**
 *chúng ta không thể thay đổi nhưng chúng ta có thể cấp quyền trên nó
 * @author Viet Hung
 */
public class Folder implements IFolder {
     public void coppy()
 {
       //truy cập vào thư mục và thực hiện sao chép
  System.out.println("thực hiện coppy trên thư mục");
 }

}
```

## -Folderproxy.java

```
package proxyDP;

/**
 *cung cấp  phép cho lớp  Folder. Nó kiểm tra username  và password  và nếu kết hợp, sau đó chỉ có nó cho phép truy cập vào thư mục.
```

```
 * @author Viet Hung
 */
public class FolderProxy implements IFolder {
    Folder folder;
 User user;

 public FolderProxy(User user) {
  this.user = user;
 }

   /**
    *
    */
   @Override
 public void coppy() {

  if(user.getUserName().equalsIgnoreCase("admin") &&
  user.getPassword().equalsIgnoreCase("admin"))
  {
   folder=new Folder();
   folder.coppy();
  }
  else
  {
   System.out.println("bạn không thể làm gì");
  }
     }

}
```

## -User.java

```
package proxyDP;

/**
 *
 * @author Viet Hưng
 */
public class User {
    String userName;
 String password;
```

```
 public User(String userName, String password) {
  this.userName = userName;
  this.password = password;
 }

 public String getUserName() {
  return userName;
 }
 public String getPassword() {
  return password;
 }


}
```

-main.java

```
package proxyDP;

/**
 *
 * @author Viet Hung
 */
public class main {
    public static void main(String[] args) {

  // Khi bạn click vào thư mục, hệ thống sẽ hỏi userName và mật khẩu.
  //và hệ thống sẽ tạo đối tượng người dùng này

  //nếu cung cấp đúng mật khẩu
  User user=new User("admin","admin");
  FolderProxy folderProxy=new FolderProxy(user);
  System.out.println("Khi userName và password chính xác:");
  folderProxy.coppy();
  System.out.println("**************************************");
  // if we give wrong userName and Password
  User userWrong=new User("abc","abc");
  FolderProxy folderProxyWrong=new FolderProxy(userWrong);
  System.out.println(" Khi userName và password không chính xác:");
  folderProxyWrong.coppy();
 }
```

}

## IV. Phân công:

Phạm Đức Hiển: State pattern
Nguyễn Văn Hãnh: Memento pattern.
Đỗ Việt Hưng: Proxy pattern.

**--- END ---**

## Overview

**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat. Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper.**

## Goals

1. **Lorem ipsum dolor sit amet, consectetuer adipiscing elit**
2. **Sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.**

## Specifications

**Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer possim assum. Typi non habent claritatem insitam; est usus legentis in iis qui facit eorum claritatem. Investigationes demonstraverunt lectores legere me lius quod ii legunt saepius.**

### Lorem Ipsum

**Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.**

## Milestones

### I. Lorem ipsum

**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.**

### II. Dolor sit amet

**Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.**