



ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG


Bài 11: Các nguyên lý thiết kế (tiếp)

1

1

Nội dung

- ❖ 1. Nguyên lý GRASP
- ❖ 2. Nguyên lý tri thức tối thiểu
- ❖ 3. Nguyên lý SOLID




VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

2

2

SOLID

- S – Single-responsibility principle
- O – Open-closed principle
- L – Liskov substitution principle
- I – Interface segregation principle
- D – Dependency Inversion Principle



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

3

3


3.3. Nguyên lý thay thế Liskov

Liskov substitution principle

- ❖ “Let $q(x)$ be a property provable about objects of x of type T . Then $q(y)$ should be provable for objects y of type S where S is a subtype of T ”
- ❖ Lớp con phải hoàn toàn thay thế được cho lớp cha (không thay đổi bản chất hành vi của lớp cha)

Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it

New derived classes are extending the base classes without changing their behavior



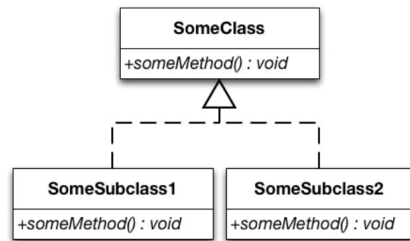
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

4

4

Nguyên lý thay thế Liskov

```
void clientMethod(SomeClass sc) {  
    ...  
    sc.someMethod();  
    ...  
}
```



5

Ví dụ – Rectangle

```
class Rectangle {  
    protected int m_width;  
    protected int m_height;  
  
    public void setWidth(int width){  
        m_width = width;  
    }  
    public void setHeight(int height){  
        m_height = height;  
    }  
    public int getWidth(){  
        return m_width;  
    }  
    public int getHeight(){  
        return m_height;  
    }  
    public int getArea(){  
        return m_width * m_height;  
    }  
}
```

6

Square

```
class Square extends Rectangle {  
    @override  
    public void setWidth(int width){  
        m_width = width;  
        m_height = width;  
    }  
    @override  
    public void setHeight(int height){  
        m_width = height;  
        m_height = height;  
    }  
}
```

7

Test

```
public class RectangleFactory {  
    public static Rectangle generate(){  
        return new Square(); // if we return new Rectangle(), everything is fine  
    }  
}  
  
class LspTest {  
    public static void main (String args[]) {  
        Rectangle r = RectangleFactory.generate();  
  
        r.setWidth(5);  
        r.setHeight(10);  
        // user knows that r it's a rectangle.  
        // It assumes that he's able to set the width and height as for the base class  
  
        System.out.println(r.getArea());  
        // now he's surprised to see that the area is 100 instead of 50.  
    }  
}
```

8

Giải pháp 1 (chưa tối ưu) – Shape

```
public abstract class Shape {
    protected int mHeight;
    protected int mWidth;

    public abstract int getWidth();

    public abstract void setWidth(int inWidth);

    public abstract int getHeight();

    public abstract void setHeight(int inHeight);

    public int getArea() {
        return mHeight * mWidth;
    }
}
```

9

Rectangle

```
public class Rectangle extends Shape {
    @Override
    public int getWidth() {
        return mWidth;
    }

    @Override
    public int getHeight() {
        return mHeight;
    }

    @Override
    public void setWidth(int inWidth) {
        mWidth = inWidth;
    }

    @Override
    public void setHeight(int inHeight) {
        mHeight = inHeight;
    }
}
```

10

Square

```
public class Square extends Shape {
    @Override
    public int getWidth() {
        return mWidth;
    }
    @Override
    public void setWidth(int inWidth) {
        SetWidthAndHeight(inWidth);
    }
    @Override
    public int getHeight() {
        return mHeight;
    }
    @Override
    public void setHeight(int inHeight) {
        SetWidthAndHeight(inHeight);
    }
    private void setWidthAndHeight(int inValue) {
        mHeight = inValue;
        mWidth = inValue;
    }
}
```

11

Test

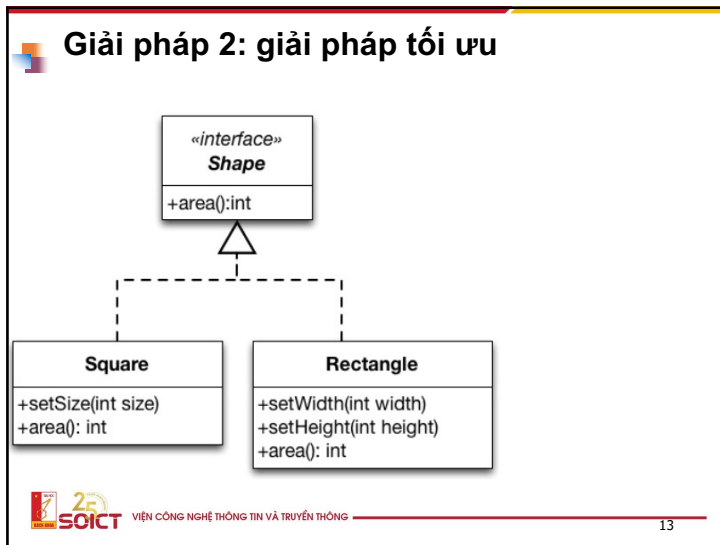
```
public class ShapeFactory {
    public static Shape generate(){
        return new Square();
    }
}

class LspTest {
    public static void main (String args[]) {
        Shape s = ShapeFactory.generate();

        s.setWidth(5);
        s.setHeight(10);

        System.out.println(r.getArea());
    }
}
```

12



13

Ví dụ 2 – Project

```

public class Project {
    public ArrayList<ProjectFile> projectFiles;

    public void loadAllFiles() {
        for (ProjectFile file: projectFiles) {
            file.loadFileData();
        }
    }

    public void saveAllFiles() {
        for (ProjectFile file: projectFiles) {
            file.saveFileData();
        }
    }
}
  
```

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

14

ProjectFile

```

public class ProjectFile {
    public string filePath;

    public byte[] fileData;

    public void loadFileData() {
        // Retrieve FileData from disk
    }

    public void saveFileData() {
        // Write FileData to disk
    }
}
  
```

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

15

ReadOnlyFile

```

public class ReadOnlyFile extends ProjectFile {
    @Override
    public void saveFileData() throws new
    InvalidOPEException {
        throw new InvalidOPEException();
    }
}
  
```

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

16

Project

```
public class Project {  
    public ArrayList<ProjectFile> projectFiles;  
  
    public void loadAllFiles() {  
        for (ProjectFile file: projectFiles) {  
            file.loadFileData();  
        }  
    }  
  
    public void saveAllFiles() {  
        for (ProjectFile file: projectFiles) {  
            if (!file instanceof ReadOnlyFile)  
                file.saveFileData();  
        }  
    }  
}
```

17

Project

```
public class Project {  
    public ArrayList<ProjectFile> allFiles;  
    public ArrayList<WritableFile> writableFiles ;  
  
    public void loadAllFiles() {  
        for (ProjectFile file: allFiles) {  
            file.loadFileData();  
        }  
    }  
  
    public void saveAllFiles() {  
        for (ProjectFile file: writableFiles) {  
            file.saveFileData();  
        }  
    }  
}
```

18

ProjectFile

```
public class ProjectFile {  
    public string filePath;  
  
    public byte[] fileData;  
  
    public void loadFileData() {  
        // Retrieve FileData from disk  
    }  
}
```

19

WritableFile

```
public class WritableFile extends ProjectFile {  
    public void saveFileData() {  
        // Write FileData to disk  
    }  
}
```

20

Discussion

- ❖ Does **method overriding** break the Liskov substitution principle?

21

Ví dụ

```
public class Report{  
    private Foo foo;  
    public String toString(){  
        return "";  
    }  
}
```

Ba lớp con
1. HTMLReport
2. XMLReport
3. TextReport

❖ Hợp đồng:

- toString() trả về một chuỗi, là biểu diễn của Foo ở một format nào đó (và trả về chuỗi rỗng nếu format chưa xác định).
- toString() không làm thay đổi đối tượng Report
- toString() không tung ra một ngoại lệ (Exception) mới

22

4. Nguyên lý phân tách giao diện Interface Segregation Principle

- ❖ Một giao diện (interface) không nên chứa các phương thức mà lớp thực thi không cần đến (Client không nên bị ép phụ thuộc vào các phương thức mà chúng không sử dụng)
- ❖ Một interface đa năng (“Fat interface”) nên được chia tách thành các interface nhỏ hơn, có nhiệm vụ cụ thể

23

Ví dụ – Toy

```
public interface Toy {  
    void setPrice(double price);  
    void setColor(String color);  
    void move();  
    void fly();  
}
```

24

ToyHouse

```
public class ToyHouse implements Toy {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }
    @Override
    public void setColor(String color) {
        this.color = color;
    }
    @Override
    public void move(){}
    @Override
    public void fly(){}
}
```

25

25

Mã nguồn sửa đổi

```
public interface Toy {
    void setPrice(double price);
    void setColor(String color);
}

public interface Movable {
    void move();
}

public interface Flyable {
    void fly();
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

26

26

ToyHouse

```
public class ToyHouse implements Toy {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }
    @Override
    public void setColor(String color) {
        this.color = color;
    }
    @Override
    public String toString(){
        return "ToyHouse: Toy house- Price: "+price+" Color: "+color;
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

27

27

```
public class ToyCar implements Toy, Movable {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }
    @Override
    public void setColor(String color) {
        this.color = color;
    }
    @Override
    public void move(){
        System.out.println("ToyCar: Start moving car.");
    }
    @Override
    public String toString(){
        return "ToyCar: Moveable Toy car- Price: "+price+" Color: "+color;
    }
}
```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

28

28

```

public class ToyPlane implements Toy, Movable, Flyable {
    double price;
    String color;

    @Override
    public void setPrice(double price) {
        this.price = price;
    }

    @Override
    public void setColor(String color) {
        this.color = color;
    }

    @Override
    public void move() {
        System.out.println("ToyPlane: Start moving plane.");
    }

    @Override
    public void fly() {
        System.out.println("ToyPlane: Start flying plane.");
    }

    @Override
    public String toString() {
        return "ToyPlane: Moveable and flyable toy plane- Price: "+price+" Color: "+color;
    }
}

```

29

```

public class ToyBuilder {
    public static ToyHouse buildToyHouse(){
        ToyHouse toyHouse=new ToyHouse();
        toyHouse.setPrice(15.00);
        toyHouse.setColor("green");
        return toyHouse;
    }

    public static ToyCar buildToyCar(){
        ToyCar toyCar=new ToyCar();
        toyCar.setPrice(25.00);
        toyCar.setColor("red");
        toyCar.move();
        return toyCar;
    }

    public static ToyPlane buildToyPlane(){
        ToyPlane toyPlane=new ToyPlane();
        toyPlane.setPrice(125.00);
        toyPlane.setColor("white");
        toyPlane.move();
        toyPlane.fly();
        return toyPlane;
    }
}

```

30

Nguyên lý phân tách giao diện và nguyên lý một nhiệm vụ

- ❖ Cùng mục đích: đảm bảo tính chất tập trung, nhỏ gọn, kết dính cao (highly cohesive) cho các thành phần phần mềm
- ❖ Nguyên lý một nhiệm vụ áp dụng cho lớp
- ❖ Nguyên lý phân tách giao diện áp dụng cho giao diện

31

Ví dụ 2 – RestaurantInterface

```

public interface RestaurantInterface {
    public void acceptOnlineOrder();
    public void takeTelephoneOrder();
    public void payOnline();
    public void walkInCustomerOrder();
    public void payInPerson();
}

```

32


```

public class OnlineClientImpl implements RestaurantInterface {
    @Override
    public void acceptOnlineOrder() {
        // logic for placing online order
    }

    @Override
    public void takeTelephoneOrder() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }

    @Override
    public void payOnline() {
        // logic for paying online
    }

    @Override
    public void walkInCustomerOrder() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }

    @Override
    public void payInPerson() { // Not Applicable for Online Order
        throw new UnsupportedOperationException();
    }
}

```

33

33

3.5. Nguyên lý đảo ngược sự phụ thuộc Dependency Inversion Principle

- ❖ Các ứng dụng thường thiết kế theo hướng top-down (các module mức cao được thiết kế dựa trên các module mức thấp). Hậu quả là module mức cao bị phụ thuộc vào module mức thấp
- ❖ Các modules cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả hai nên phụ thuộc vào sự trừu tượng hóa

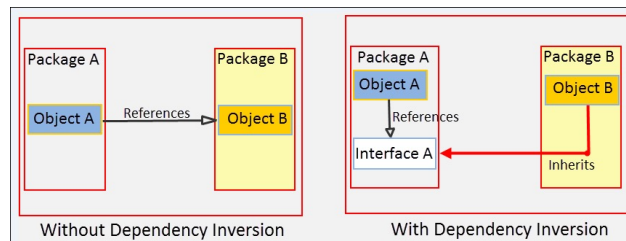


VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

35

35

3.5. Nguyên lý đảo ngược sự phụ thuộc



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

36

36

Ví dụ – LightBulb

```

public class LightBulb {
    public void turnOn() {
        System.out.println("LightBulb: Bulb turned on...");
    }

    public void turnOff() {
        System.out.println("LightBulb: Bulb turned off...");
    }
}

```



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

37

37

ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public LightBulb lightBulb;  
    public boolean on;  
    public ElectricPowerSwitch(LightBulb lightBulb) {  
        this.lightBulb = lightBulb;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            lightBulb.turnOff();  
            this.on = false;  
        } else {  
            lightBulb.turnOn();  
            this.on = true;  
        }  
    }  
}
```



38

Interface ISwitchable

```
public interface ISwitchable {  
    public void turnOn();  
    public void turnOff();  
}
```



39

ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```



40

LightBulb

```
public class LightBulb implements ISwitchable {  
    @Override  
    public void turnOn() {  
        System.out.println("LightBulb: Bulb turned on...");  
    }  
  
    @Override  
    public void turnOff() {  
        System.out.println("LightBulb: Bulb turned off...");  
    }  
}
```



41

Fan

```
public class Fan implements ISwitchable {  
    @Override  
    public void turnOn() {  
        System.out.println("Fan: Fan turned on...");  
    }  
  
    @Override  
    public void turnOff() {  
        System.out.println("Fan: Fan turned off...");  
    }  
}
```



42

ElectricPowerSwitchTest

```
public class ElectricPowerSwitchTest {  
  
    @Test  
    public void testPress() throws Exception {  
        ISwitchable switchableBulb=new LightBulb();  
        ElectricPowerSwitch bulbPowerSwitch =  
            new ElectricPowerSwitch(switchableBulb);  
        bulbPowerSwitch.press();  
        bulbPowerSwitch.press();  
  
        ISwitchable switchableFan=new Fan();  
        ElectricPowerSwitch fanPowerSwitch =  
            new ElectricPowerSwitch(switchableFan);  
        fanPowerSwitch.press();  
        fanPowerSwitch.press();  
    }  
}
```



43

ElectricPowerSwitch

```
public class ElectricPowerSwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```

Any problem?



44

ISwitch

```
public interface ISwitch {  
    boolean isOn();  
    void press();  
}
```



45

ElectricPowerSwitch

```
public class ElectricPowerSwitch implements ISwitch {  
    public ISwitchable client;  
    public boolean on;  
    public ElectricPowerSwitch(ISwitchable client) {  
        this.client = client;  
        this.on = false;  
    }  
    public boolean isOn() {  
        return this.on;  
    }  
    public void press(){  
        boolean checkOn = isOn();  
        if (checkOn) {  
            client.turnOff();  
            this.on = false;  
        } else {  
            client.turnOn();  
            this.on = true;  
        }  
    }  
}
```

