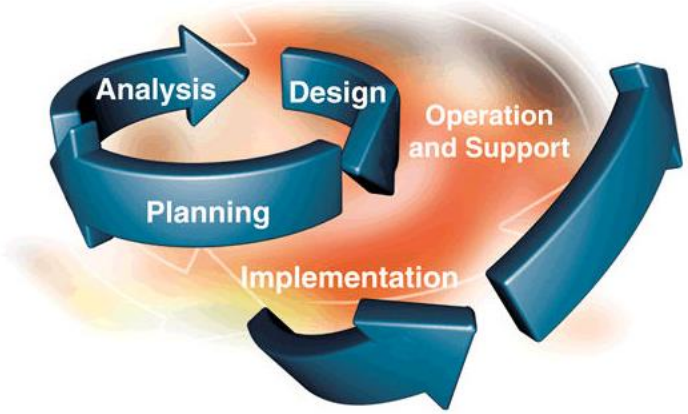
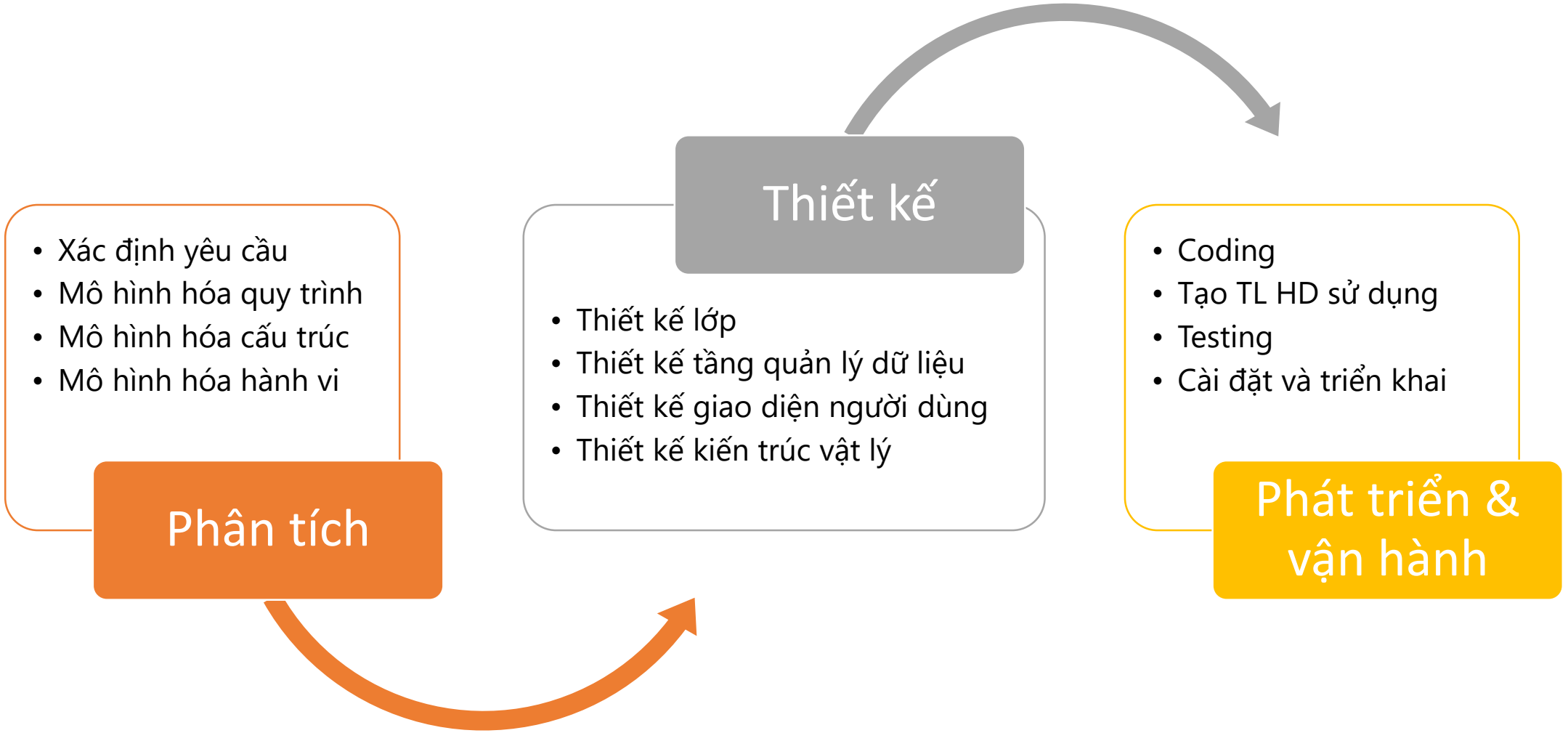
THIẾT KẾ LỚP

Giảng viên: Cao Thị Nhâm

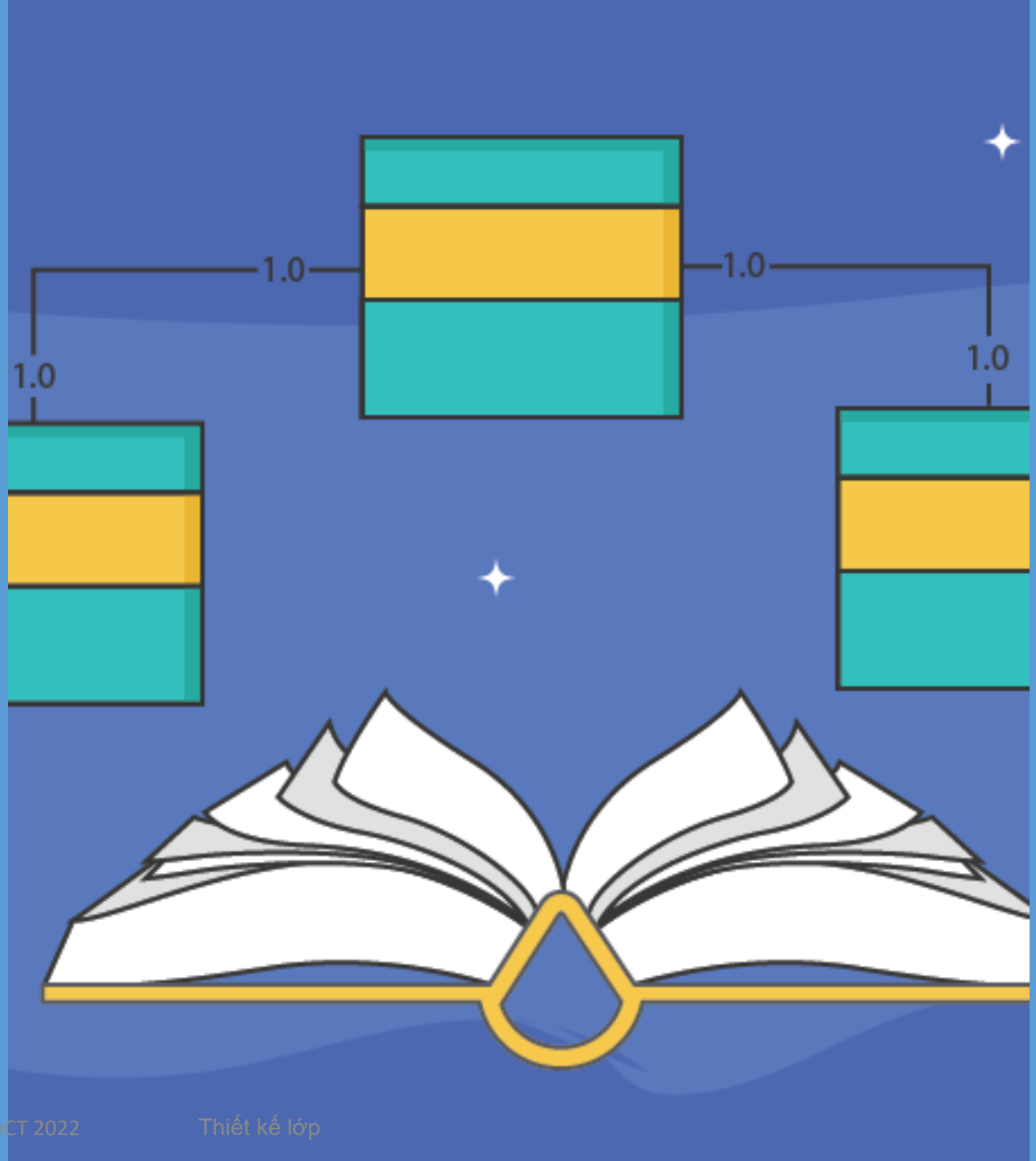


BỨC TRANH TỔNG THỂ



Nội dung chính

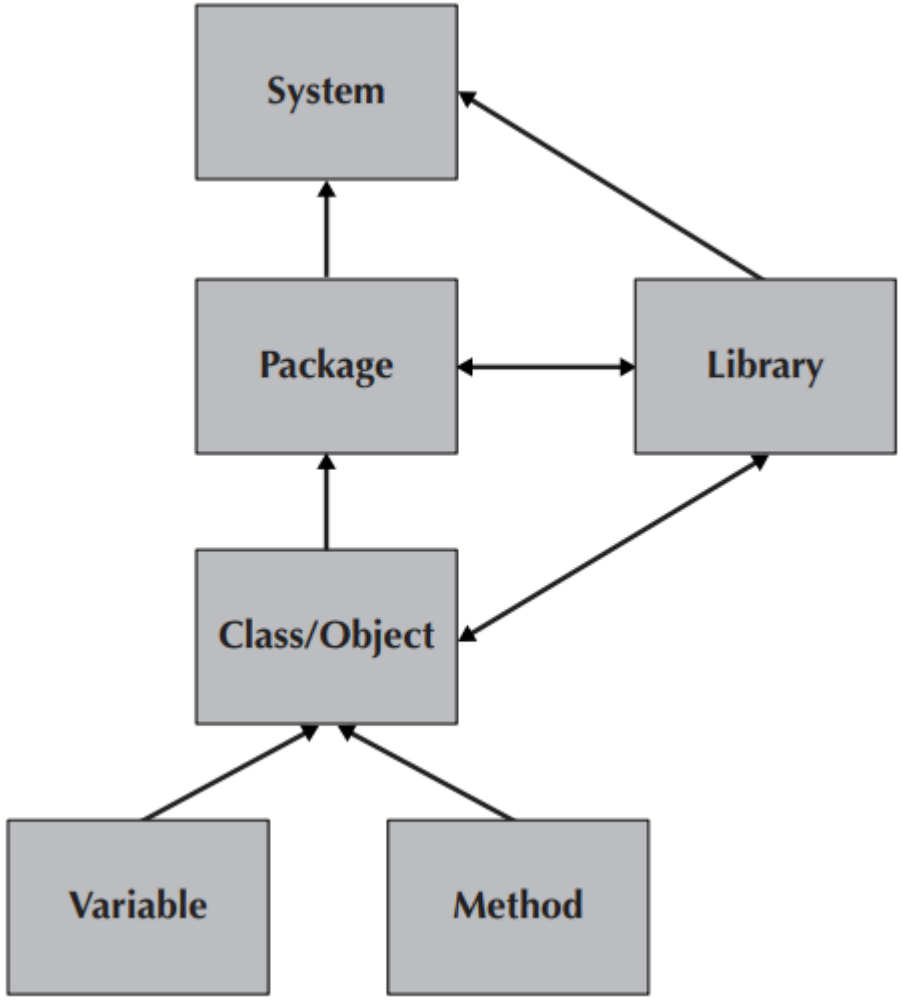
- Thiết kế lớp
- Thiết kế tầng quản lý dữ liệu
- Thiết kế giao diện người dùng
- Thiết kế kiến trúc vật lý



THIẾT KẾ LỚP

- Nguyên lý SOLID
- Thiết kế lớp
- Đặc tả phương thứ
- Thiết kế ràng buộc

Cấu trúc hệ thống



NHẮC LẠI

- Lớp
- Đối tượng
- Hành vi
- Phương thức
- Đặc điểm của OOP:
  - Đóng gói
  - Đa hình
  - Thừa kế
  - Trừu tượng

Các nguyên lý thiết kế

A good design is one that balances trade-offs to minimize the total cost of the system over its entire lifetime.

Coad and Yourdon

- Nguyên lý **SOLID**
  - **S**ingle responsibility principle
  - **O**pen/closed principle
  - **L**iskov Substitution Principle
  - **I**nterface Segregation Principle
  - **D**ependency inversion principle

Single responsibility

- **Một class chỉ nên giữ một trách nhiệm duy nhất**

→ Dễ hiểu, dễ bảo trì, dễ sửa đổi

Single responsibility...

```java
class Invoice {
    private Marker marker;
    private int quantity;

    public Invoice(Marker marker, int quantity) {
        this.marker = marker;
        this.quantity = quantity;
    }

    public int calculateTotal() {
        return marker.price * this.quantity;
    }

    public void printInvoice() {
        // printing implementation
    }

    public void saveToDb() {
        // save to database implementation
    }
}
```

```java
class Invoice {
    private Marker marker;
    private int quantity;

    public Invoice(Marker marker, int quantity) {
        this.marker = marker;
        this.quantity = quantity;
    }

    public int calculateTotal() {
        return marker.price * this.quantity;
    }
}
```

```java
class InvoicePrinter {
    private Invoice invoice;

    public InvoicePrinter(Invoice invoice) {
        this.invoice = invoice;
    }

    public void printInvoice() {
        // printing implementation
    }
}
```

```java
class InvoiceDao {
    private Invoice invoice;

    public InvoiceDao(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToDb() {
        // save to database implementation
    }
}
```

Open/closed

- **"Open for extension but closed for modification" → Có thể mở rộng một class nhưng không được sửa đổi bên trong nó**

→ Dễ mở rộng, dễ bảo trì

- Mỗi khi muốn thêm chức năng cho chương trình, nên viết class mới mở rộng class cũ (bằng cách sử dụng inheritance hoặc interface), không nên sửa đổi class cũ.

Open/closed...

```
class InvoiceDao {
    private Invoice invoice;

    public InvoiceDao(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToDb() {
        // save to database implementation
    }
}
```

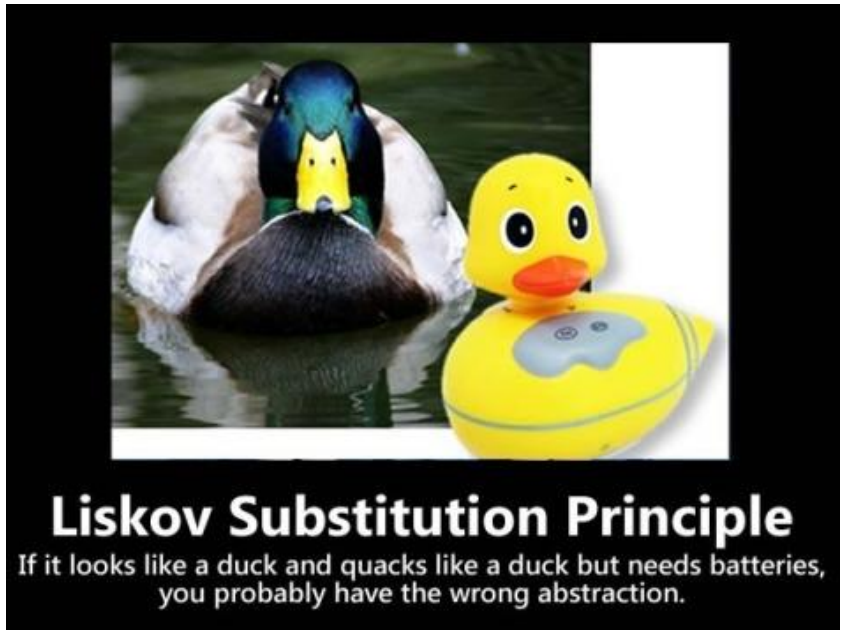
```
interface InvoiceDao {
    public void save(Invoice invoice);
}

class DatabaseInvoiceDao implements InvoiceDao {
    @Override
    public void save(Invoice invoice) {
        // save to database implementation
    }
}

class FileInvoiceDao implements InvoiceDao {
    @Override
    public void save(Invoice invoice) {
        // save to file implementation
    }
}
```

Cần lưu Invoice dưới dạng file?

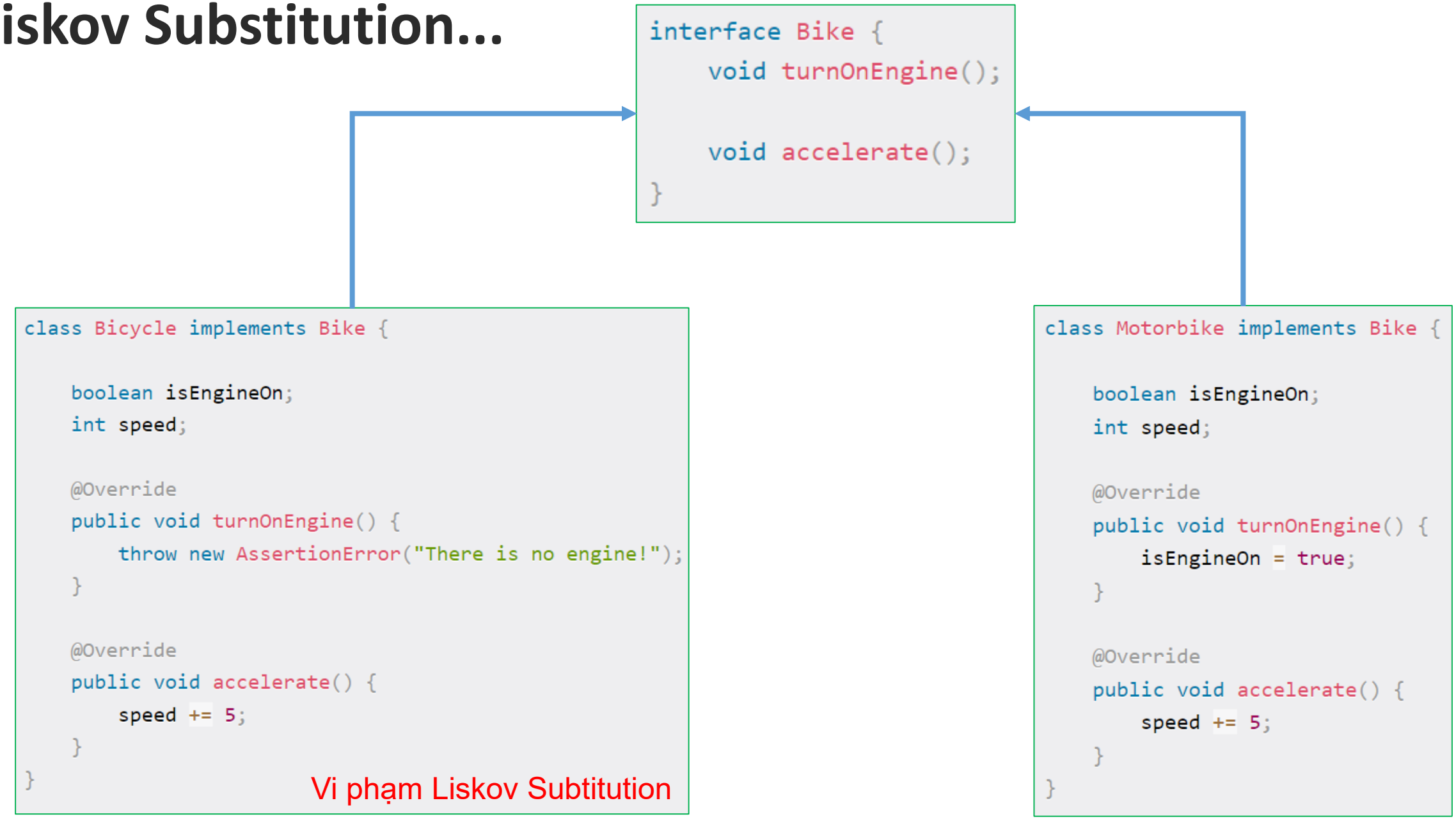
Liskov Substitution

- Lớp con nên hoạt động giống như lớp cơ sở (cha) của nó trong mọi ngữ cảnh

→ Đồng nhất trong xử lý, dễ dự đoán



Liskov Substitution...



```java
interface Bike {
    void turnOnEngine();

    void accelerate();
}
```

```java
class Bicycle implements Bike {

    boolean isEngineOn;
    int speed;

    @Override
    public void turnOnEngine() {
        throw new AssertionError("There is no engine!");
    }

    @Override
    public void accelerate() {
        speed += 5;
    }
}
```

Vi phạm Liskov Subtitution

```java
class Motorbike implements Bike {

    boolean isEngineOn;
    int speed;

    @Override
    public void turnOnEngine() {
        isEngineOn = true;
    }

    @Override
    public void accelerate() {
        speed += 5;
    }
}
```

Interface Segregation

- Thay vì dùng 1 interface lớn, nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể

→ Linh hoạt, dễ bảo trì

Interface Segregation...

```java
interface Vehicle {
    void startEngine();
    void stopEngine();
    void drive();
    void fly();
}
```

```java
class Car implements Vehicle {

    @Override
    public void startEngine() {
        // implementation
    }

    @Override
    public void stopEngine() {
        // implementation
    }

    @Override
    public void drive() {
        // implementation
    }

    @Override
    public void fly() {
        throw new UnsupportedOperationException("This vehicle cannot fly.");
    }
}
```

Vi phạm Interface Segregation

Interface Segregation...

```
interface Drivable {
    void startEngine();
    void stopEngine();
    void drive();
}

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

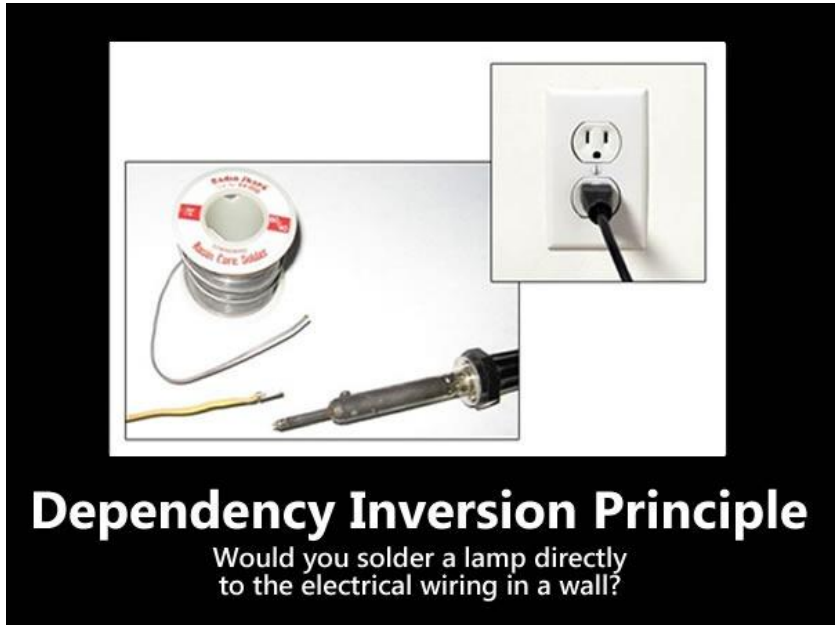
```
class Car implements Drivable {

    @Override
    public void startEngine() {
        // implementation
    }

    @Override
    public void stopEngine() {
        // implementation
    }

    @Override
    public void drive() {
        // implementation
    }
}
```

```
class Airplane implements Drivable, Flyable {

    @Override
    public void startEngine() {
        // implementation
    }

    @Override
    public void stopEngine() {
        // implementation
    }

    @Override
    public void drive() {
        // implementation
    }

    @Override
    public void fly() {
        // implementation
    }
}
```

Dependency inversion

- Các module cấp cao không nên phụ thuộc vào các modules cấp thấp. Cả 2 nên phụ thuộc vào abstraction

→ Dễ bảo trì, dễ kiểm thử, dễ mở rộng



Dependency inversion

```
class WeatherTracker {
    private String currentConditions;
    private Emailer emailer;

    public WeatherTracker() {
        this.emailer = new Emailer();
    }

    public void setCurrentConditions(String weatherDescription) {
        this.currentConditions = weatherDescription;
        if (weatherDescription == "rainy") {
            emailer.sendEmail("It is rainy");
        }
    }
}

class Emailer {
    public void sendEmail(String message) {
        System.out.println("Email sent: " + message);
    }
}
```

Vi phạm Dependency inversion

```
interface Notifier {
    public void alertWeatherConditions(String weatherDescription);
}

class WeatherTracker {
    private String currentConditions;
    private Notifier notifier;

    public WeatherTracker(Notifier notifier) {
        this.notifier = notifier;
    }

    public void setCurrentConditions(String weatherDescription) {
        this.currentConditions = weatherDescription;
        if (weatherDescription == "rainy") {
            notifier.alertWeatherConditions("It is rainy");
        }
    }
}
```

```
class Emailer implements Notifier {
    public void alertWeatherConditions(String weatherDescription) {
        System.out.println("Email sent: " + weatherDescription);
    }
}

class SMS implements Notifier {
    public void alertWeatherConditions(String weatherDescription) {
        System.out.println("SMS sent: " + weatherDescription);
    }
}
```

Công việc cần làm

Thiết kế lớp

Đặc tả phương thức

Thiết kế ràng buộc

Review các thiết kế

Thiết kế lớp


1
Thêm đặc tả (thuộc tính, phương thức)
2
Xác định các lớp có thể tái sử dụng
3
Tái cấu trúc lớp & tối ưu thiết kế
4
Thiết kế lớp vật lý

Thiết kế lớp (tiếp)

Đặc tả phương thức

- Là một tài liệu thể hiện cách thức thực hiện của phương thức

→người lập trình dựa trên tài liệu này để code

- Nội dung đặc tả gồm 4 phần chính:
  - Thông tin tổng quát
  - Sự kiện kích hoạt
  - Tham số
  - Đặc tả thuật toán

Đặc tả phương thức (tiếp)

Method name:	Class name:	Programmer:
Events:		
Inputs	Name	Data type
Outputs		
Algorithm specification		
Notes		

Thông tin tổng quát

Sự kiện kích hoạt

Tham số

Đặc tả thuật toán

Các ghi chú khác

Thiết kế ràng buộc

- Các loại ràng buộc
  - Điều kiện tiên quyết (Pre-condition)
    - Ràng buộc cần đạt được để một phương thức có thể thực thi
    - Ví dụ: tham số đầu vào của phương thức Tìm ước số của hai số phải có giá trị nguyên
  - Hậu điều kiện (Post-condition)
    - Ràng buộc cần đạt được sau khi thực thi một phương thức
    - Ví dụ: Điểm số của sinh viên phải được lưu vào cơ sở dữ liệu sau khi kết thúc phương thức
  - Invariant
    - Ràng buộc áp dụng cho các thuộc tính của lớp
    - Ví dụ: Thuộc tính Tuổi phải có giá trị nguyên nằm trong khoảng 0 - 120


Q&A