TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

🙧🕮🙥

A picture containing logo

Description automatically generated

Báo cáo Tìm hiểu về Single Responsibility Principle (SRP) và Don't Repeat Yourself (DRY) Principle

Môn: Thiết kế phần mềm

GVHD: Trần Duy Thảo

*Thành phố Hồ Chí Minh, tháng 2, năm 2025*

# Giới thiệu

Lập trình phần mềm đòi hỏi việc tuân thủ các nguyên tắc tốt nhất để đảm bảo tính bảo trì, mở rộng và dễ đọc. Hai nguyên tắc quan trọng giúp đạt được điều này là Single Responsibility Principle (SRP) và Don't Repeat Yourself (DRY) Principle. Báo cáo này phân tích hai nguyên tắc này, tầm quan trọng, cách áp dụng và mối quan hệ giữa chúng.

# Single Responsibility Principle (SRP)

Định nghĩa

Single Responsibility Principle (SRP) quy định rằng một class chỉ nên có một lý do duy nhất để thay đổi. Điều này có nghĩa là một class chỉ nên đảm nhiệm một chức năng cụ thể trong hệ thống phần mềm.

Tầm quan trọng

* Cải thiện khả năng bảo trì bằng cách đảm bảo rằng sự thay đổi của một chức năng không ảnh hưởng đến chức năng khác.
* Nâng cao khả năng kiểm thử bằng cách cô lập từng chức năng.
* Giúp code dễ hiểu và dễ chỉnh sửa.
* Giảm sự phụ thuộc giữa các thành phần trong hệ thống.

Ví dụ thực tế

Ví dụ 1: Vi phạm SRP

class Report {

public void generateReport() {

// Logic tạo báo cáo

}

public void printReport() {

// Logic in báo cáo

}

public void saveReportToDatabase() {

// Logic lưu báo cáo vào CSDL

}

}

Lớp Report thực hiện ba nhiệm vụ khác nhau, điều này vi phạm SRP.

Ví dụ 2: Tuân thủ SRP

class ReportGenerator {

public void generateReport() {

// Logic tạo báo cáo

}

}

class ReportPrinter {

public void printReport() {

// Logic in báo cáo

}

}

class ReportSaver {

public void saveReportToDatabase() {

// Logic lưu báo cáo vào CSDL

}

}

Bằng cách chia nhỏ trách nhiệm, mỗi class đảm nhiệm một chức năng duy nhất, giúp code dễ bảo trì hơn.

Cách áp dụng SRP trong lập trình

* Xác định các nhiệm vụ khác nhau trong một class.
* Tách riêng các chức năng thành các class khác nhau.
* Sử dụng Dependency Injection để quản lý sự phụ thuộc hiệu quả hơn.

# Don't Repeat Yourself (DRY) Principle

Định nghĩa

Don't Repeat Yourself (DRY) Principle quy định rằng không nên lặp lại cùng một đoạn code nhiều lần. Thay vì lặp lại, ta nên trừu tượng hóa và tái sử dụng các thành phần chung.

Tầm quan trọng

* Giảm sự dư thừa trong code, giúp giảm lỗi và dễ cập nhật hơn.
* Cải thiện tính tái sử dụng của mã nguồn.
* Nâng cao khả năng bảo trì, vì sửa đổi ở một nơi có thể áp dụng cho toàn bộ hệ thống.

Ví dụ thực tế

Ví dụ 1: Vi phạm DRY

class Employee {

public double calculateBonus(double salary) {

return salary \* 0.10;

}

}

class Manager {

public double calculateBonus(double salary) {

return salary \* 0.20;

}

}

Logic tính tiền thưởng bị lặp lại trong hai class, gây khó khăn trong việc bảo trì.

Ví dụ 2: Tuân thủ DRY

class BonusCalculator {

public static double calculateBonus(double salary, double rate) {

return salary \* rate;

}

}

class Employee {

public double getBonus(double salary) {

return BonusCalculator.calculateBonus(salary, 0.10);

}

}

class Manager {

public double getBonus(double salary) {

return BonusCalculator.calculateBonus(salary, 0.20);

}

}

Ở đây, logic tính tiền thưởng được tập trung tại một chỗ, giúp dễ bảo trì hơn.

Cách áp dụng DRY trong lập trình

* Xác định logic trùng lặp và tách chúng thành methods hoặc utility classes.
* Sử dụng inheritance và composition để tái sử dụng mã nguồn.
* Áp dụng các design patterns như Factory, Singleton, Strategy.

# Mối quan hệ giữa SRP và DRY

SRP và DRY có mối liên hệ chặt chẽ:

* SRP giúp đảm bảo rằng mỗi class chỉ thực hiện một nhiệm vụ duy nhất, điều này giúp giảm sự trùng lặp logic (hỗ trợ DRY).
* DRY khuyến khích tái sử dụng mã nguồn, phù hợp với mục tiêu của SRP trong việc tách biệt các trách nhiệm.
* Khi kết hợp SRP và DRY, ta có được mã nguồn dễ bảo trì, mở rộng và sạch sẽ.

# Thách thức và Best Practices

Thách thức

* Quá tải thiết kế: Áp dụng quá mức có thể dẫn đến quá nhiều class nhỏ.
* Cân bằng abstraction: Quá abstraction có thể làm khó khăn khi debug.
* Tích hợp legacy code: Refactor mã cũ để tuân theo SRP và DRY có thể phức tạp.

Best Practices

* Refactor mã nguồn từng bước thay vì thay đổi toàn bộ một cách đột ngột.
* Đặt tên class và method có ý nghĩa để làm rõ trách nhiệm.
* Sử dụng các design patterns để cân bằng giữa tái sử dụng và bảo trì.
* Thực hiện code review để phát hiện vi phạm SRP và DRY sớm.

# Kết luận

Single Responsibility Principle (SRP) và Don't Repeat Yourself (DRY) Principle là hai nguyên tắc quan trọng giúp viết mã nguồn sạch, dễ bảo trì và mở rộng. SRP đảm bảo rằng mỗi class có một trách nhiệm duy nhất, giúp hệ thống trở nên dễ hiểu hơn. DRY giúp loại bỏ sự dư thừa, nâng cao hiệu suất và tính nhất quán trong mã nguồn.