Chương 1

Đặt vấn đề

Chương 2

Kiến thức cơ sở

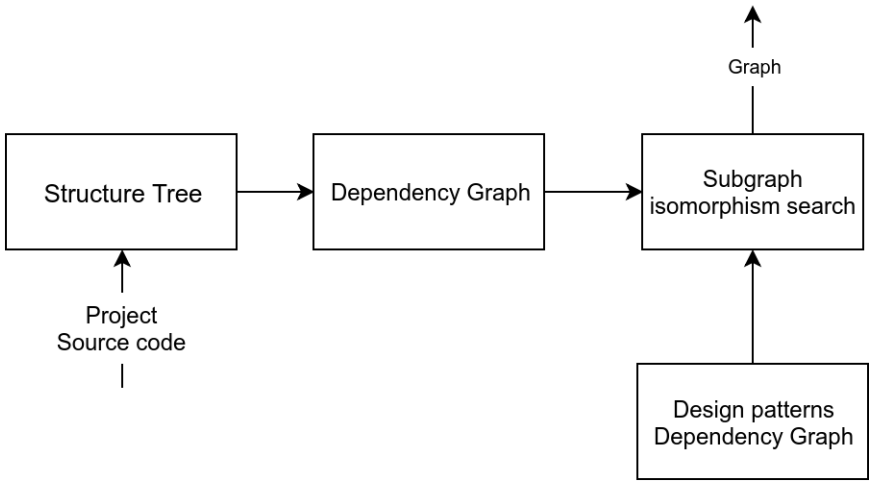
Chương 3

Phương pháp kiểm tra sự tuân thủ mẫu thiết kế cho dự án sử dụng Java

Mẫu thiết kế là tập hợp các luật nhằm mô tả cách giải quyết một vấn đề trong thiết kế có thể là vấn đề lặp lại nhiều lần trong dự án. Với những dự án công nghệ thông tin nói chung và dự án Java nói riêng. Ở các mẫu thiết kế hướng đối tượng, thường thể hiện mối quan hệ giữa các lớp, các đối tượng với nhau.

Phương pháp ở đây dựa trên phấn tích tĩnh mã nguồn, bởi vì việc phân tích mã nguồn tĩnh đem lại độ chính xác tốt và quá trình phân tích không bắt buộc mã nguồn có thể thực thi được. Do đó dữ liệu đầu vào có thể là một phần hay toàn bộ mã nguồn của dự án.

Hình 3.1 mô tả phương pháp kiểm tra sự tuân thủ mẫu thiết kế. Đầu tiên, dữ liệu đầu vào được tiền xử lý thành cây cấu trúc, thông qua cây cấu trúc tiến hành phân tích phụ thuộc bên trong mã nguồn, xây dựng đồ thị phụ thuộc. Phân tích đồ thị phụ thuộc của mã nguồn và đồ thị phụ thuộc của mẫu thiết kế nhằm kiểm tra sự tuân thủ mẫu thiết kế của mã nguồn.



Hình 3.1: Quá trình kiểm tra sự tuân thủ mẫu thiết kế của mã nguồn

3.1 Tiền xử lý mã nguồn Java

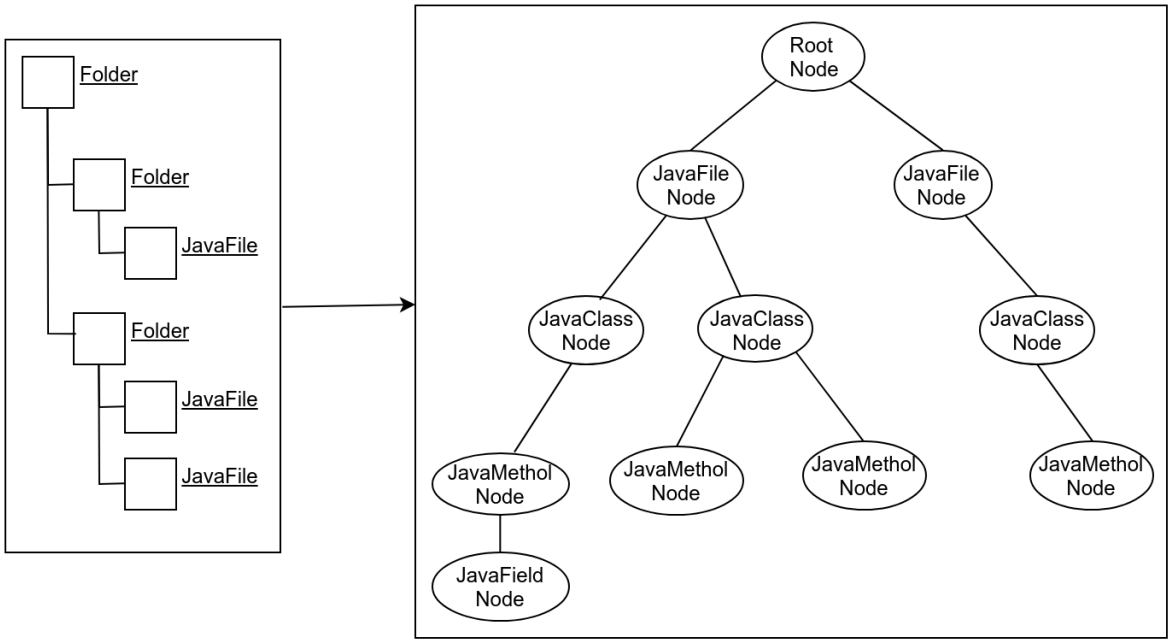
3.1.1 Xây dựng cây cấu trúc

Đối với phương pháp kiểm tra sự tuân thủ mẫu thiết kế mà khóa luận đề xuất. Cần có một kiểu dữ liệu tường minh và thể hiện được toàn bộ cấu trúc của mã nguồn, trong khi đó mã nguồn của dự án là phức tạp và chưa nhiều thông tin không được dùng tới. Nếu dùng trực tiếp mã sẽ gây khó khăn trong quá trình giải quyết bài toán và ảnh hưởng tới hiệu năng của của công cụ được xây dựng. Do đó cần tiến hành tiền

xử lý mã nguồn, xây dựng một kiểu cấu trúc dữ liệu phù hợp. Cây cấu trúc được để xuất như là một kiểu cấu trúc dữ liệu phù hợp nhất thể hiện được toàn bộ cấu trúc của mã nguồn dự án.

Định nghĩa: (*Cây cấu trúc* [1]) Là một đồ thị liên thông với $T = (N, E)$ trong đó $N = \{n_1, n_2, n_3...n_k\}$ là tập các nút trên cây đại diện cho tệp, lớp, phương thức, biến... $E = \{(e_i, e_j)|e_i \in N, e_j \in N\}$ mỗi cặp $e_i e_j$ là cặp hai đỉnh kề của đồ thị.

Mô tả phương pháp tiền xử lý mã nguồn:



Hình 3.2: Xây dựng cây cấu trúc từ mã nguồn

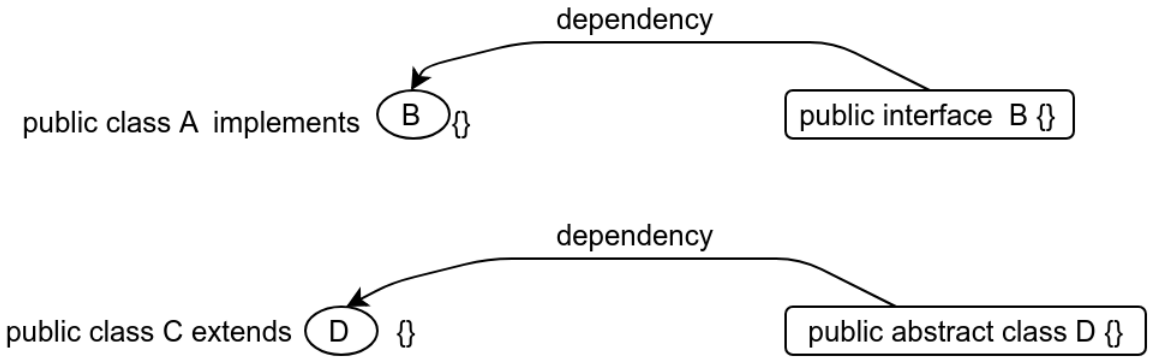
Các nút trên cây được ánh xạ về bốn loại: tệp tin (*Java*), lớp, phương thức và một loại nút thể hiện cho những định dạng còn lại. Mỗi loại nút của cây chứa những thuộc tính khác nhau và thông tin về nút cha, con của nó. Những thông tin trên mỗi nút được phân tích từ AST.

3.1.2 Xác định thuộc tính cho mỗi nút trên cây cấu trúc

Thành phần của một lớp gồm bốn phần chính: *Class type, Class dependency, Class variables, Method* . Trong đó *Class type* của một nút (class) thể hiện nút đó đóng vai trò như một: *Class, Abstract class, Template class* hay *Interface. Class depedency* ở đây ta xét tới phụ thuộc thừa kế của lớp, phụ thuộc thừa kế bao gồm hai loại: kế thừa từ một class, kế thừa từ Interface. *Method* là định nghĩa một hành vi của lớp, *Method* bao gồm các thành phần: *Local variable, Return type, Input paramater* . *Class*

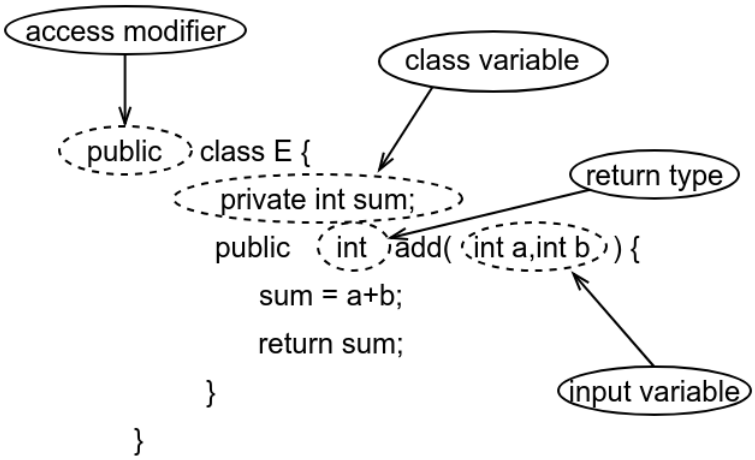
variables là biến của một lớp được khởi tạo bên ngoài các *Method*. *Local variable* là biến chỉ được khai báo và sử dụng trong phạm vi *Method*. *Return type* là kiểu dữ liệu mà phương thức sẽ trả về nếu *Return type* là kiểu *void* thì phương thức sẽ không trả về giá trị. *Input paramter* xác định kiểu giá trị đầu vào cho phương thức.

Hình 3.3 mô tả hai loại phụ thuộc kế thừa. Trong đó A là một Class thừa kế từ B là một Interface qua phương thức extend, C là một class thừa kế D qua phương thức implement với D là một abstract class.



Hình 3.3: Phụ thuộc thừa kế của lớp

Hình 3.4 Các thành phần cơ bản của *Class*. Trong đó E là một Class với Access modifer là public, Class variable là sum với kiểu giá trị int, phương thức add có kiểu trả về là int và hai biến đầu vào là a và b.



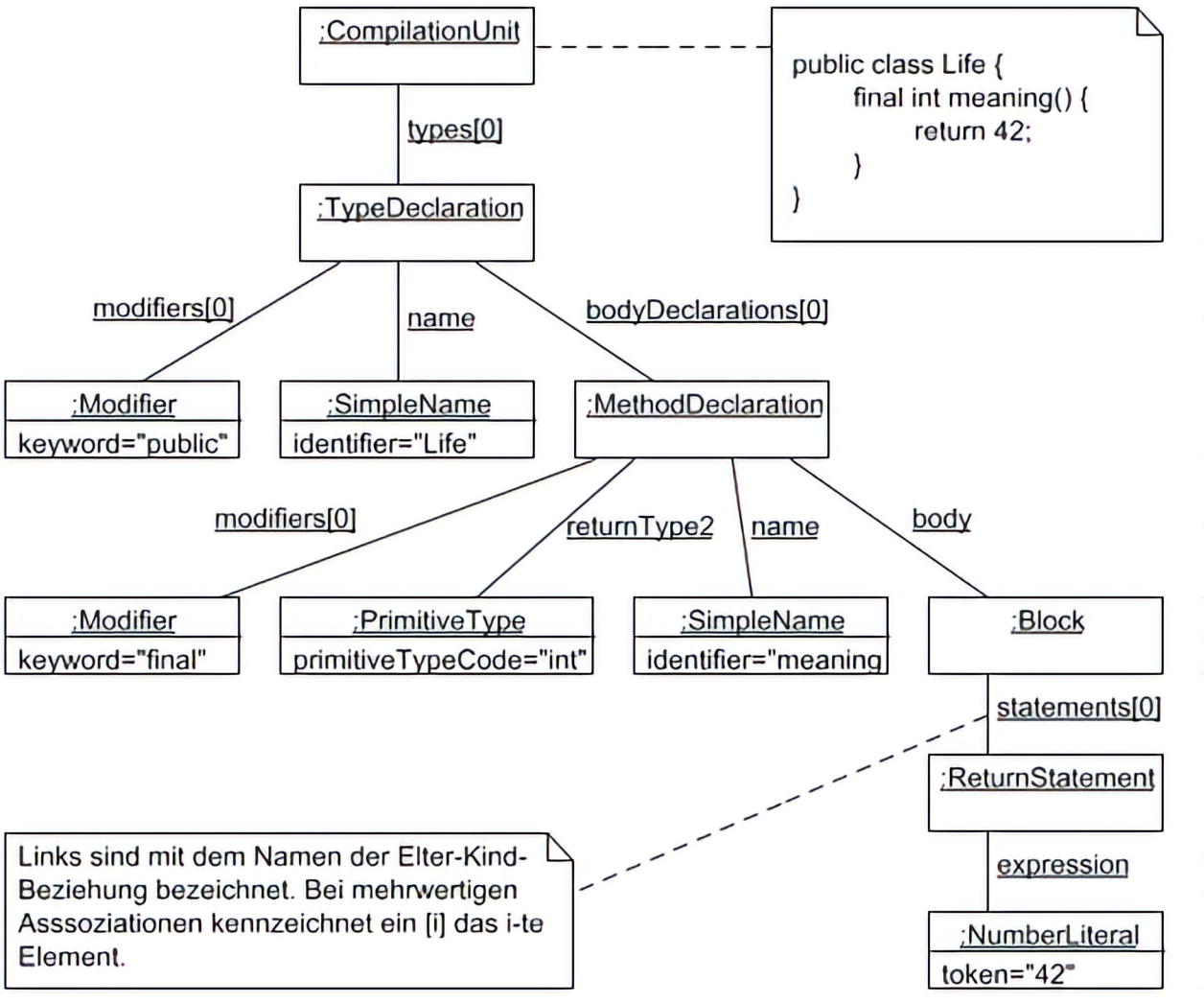
Hình 3.4: Các thành phần cơ bản trong class

Bảng 3.5 mô tả đầy đủ những thông tin cần xác định cho mỗi loại nút trên cây cấu trúc.

Node	**Properties**
Class	NameType Access modifier Extended Class Implemented Class Childrent Node: Field, Method
Method	Name NameReturn Type Access modifier Parameter Body
Field	Name Value type Access modifier

Bảng 3.1: Thuộc tính trên mỗi nút

Trích xuất các thông tin từ mã nguồn cho các nút trên cây, được thực hiện thông qua AST. Với mỗi thành phần mã nguồn, ta sử dụng JP để sinh AST tương ứng với thành phần đó từ đó trích xuất các thuộc tính cần thiết cho mỗi nút trên cây cấu trúc. Hình 3.5 mô tả một AST với một Class Java tương ứng. Trong đó một lớp Java được phân tách thành dạng cây với các nút gốc chưa các toán tử, các nút lá chứa các toán hạng. Ví dụ như return = 42, với return là một toán tử ứng với n 'ReturnStatement' và '42' là toán hạng ứng với nút lá.



Hình 3.5: Abstract syntax tree đối với Java class

3.2 Phân tích cấu trúc mã nguồn Java

Cấy cấu trúc thể hiện thể hiện chi tiết về cấu trúc của mã nguồn bao gồm các khía cạnh về tính hướng đối tượng bên trong mã nguồn. Phân tích cấu trúc mã nguồn nhằm xác định được những đặc điểm về mặt phụ thuộc giữa các thành phần mã nguồn được hình thành bới việc áp dụng những mẫu thiết kế bên trong mã nguồn. Xác định được những được những đặc nêu trên là tiền để để kiểm tra sự tuân thủ mẫu thiết kế bên trong mã nguồn.

3.2.1 Phân tích phụ thuộc giữa các thành phần trong mã nguồn

Đối với phương pháp mà khóa luận này đề xuất, việc phân tích phụ thuộc giữa các thành phân bên trong mã nguồn xoay quanh việc phân tích phụ thuộc giữa các lớp trong mã nguồn. Đối với loại phụ thuộc giữa các lớp trong mã nguồn Java bao gồm: Direct & Indirect dependency, Polymorphism dependency, Inheritance Dependency.

Polymorphism dependency: Ở đây ta xem xét hai trường hợp của loại pụụ thuộc này. Trường hợp thứ nhất khi một Class thừa kế một Interface bằng phương thức Implement. Ví dụ Class, A thừa kế một interface B, Class A sẽ thừa kế những phương thức của Interface C, tức là tại Class A những phương thức được Interface C định nghĩa sẽ được triển khai. Ngoài ra tham chiếu của Interface B có thể trỏ tới đối tượng của Class A, trong trường hợp đó đối tượng tạo được trỏ tới bởi B chỉ có thể thực hiện những phương thức mà B đã định nghĩa, nhưng phương thúc khác của A sẽ bị làm mờ đi. Trường hợp thứ hai, phụ thuộc xảy ra khi một class thừa kế một class khác thông qua phương thức extends. Ví dụ, class C thừa thế class D, lúc này ta coi D như là Class cha, với C là class con, C sẽ thừa hưởng mọi thuộc tính và phương thức của D, do đó C có thể ghi đè những phương thức của D, ngoài ra, tham chiếu của Class D có thể trỏ tới đối tượng của class C. Hình 3.6 và 3.7 mô tả ví dụ về hai trường hợp mà ta đã đề cập.

```java
public interface B {
    int add(int n1, int n2);
}
public abstract class A implements B {
    @Override
    public int add(int n1, int n2) {
        return n1 + n2;
    }
}
```

Hình 3.6: Mối quan hệ giữa một Class vớ một Interface qua phương thức Implements

```java
public class C extends D {
    @Override
    public void getAge() {
    }

    @Override
    public void getName(String name) {
        System.out.println("i'm C");
    }
}

public class Main {
    public static void main(String[] args) {
        D d = new C();
        d.getAge();
    }
}
```

Hình 3.7: Mối quan hệ giữa một Class với một Class qua phương thức extends

Inheritance Dependency: Khi một Class có được các thuộc tính và phương thức của một Class khác. Những thuộc tính và phương thức này được quản lý theo thứ tự phân cấp từ lớp con tới lớp cha, việc xử lý phân cấp được quyết định trong quán trình chương trình đang thực thi bởi JVM. Ví dụ, ta có Class D thừa kế Class E với phương thức *extends*, khi đó D sẽ thừa hưởng các phương thưc và thuộc tính của E. Trong trường hợp các phương thức và thuộc tính của D có *Access modifier* là *private*, khi đó đối tượng của Class A sẽ không thể gọi tới những thuộc tính, phương thức này. Hình 3.7 mô tả mối quan hệ thừa kết giữa hai Class Java.

```
public class E {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
}

public class D extends E {
    @Override
    public void setName(String name) {
        super.setName(name);
    }
}
```

Hình 3.8: Mối quan hệ giữa một Class vớ một Interface qua phương thức Implements

Use Dependency:

3.2.2 Xây dựng đồ thị phụ thuộc từ cây cấu trúc

3.2.3 Ví dụ minh họa

3.3 Kiểm tra sự tuân thủ mẫu thiết kế bên trong mã nguồn

Tài liệu tham khảo

Tiếng Việt

Tiếng Anh

[1] Hello latex ia

[2] Nicholas Smith, Danny van Bruggen, Federico Tomassetti JavaParser: Visited *Analyse, transform and generate your Java code base*