Getter và setter: Cách sử dụng hợp lý

Hãy theo dõi 2 đoạn code dưới đây

```
// Car1.java

public class Car1 {
  public Engine engine;
}
```

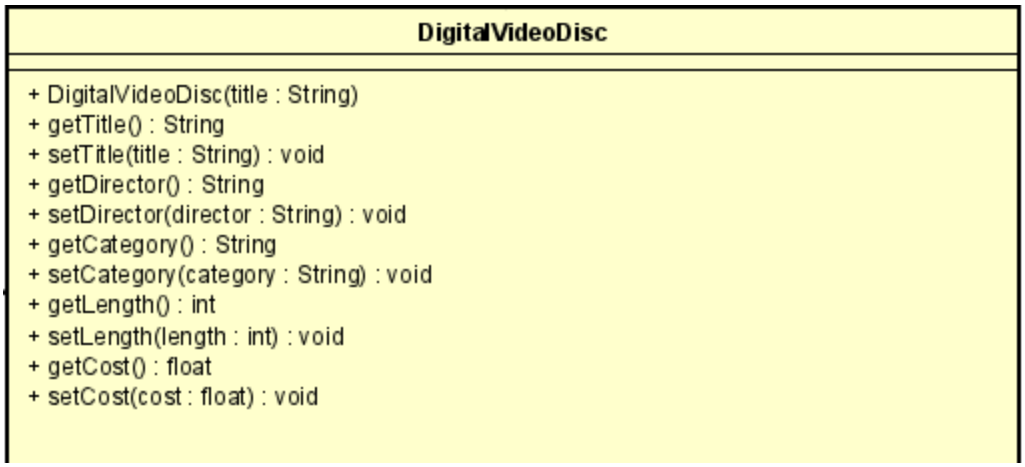
Bạn nhìn thấy thuộc tính **engine** đang có mức độ truy cập là **public** và bạn nghĩ như vậy là không an toàn, bởi ta có thể sửa thuộc tính này từ **bất kỳ nơi nào** trong project của bạn.Tôi đồng ý với suy nghĩ này của bạn.Tuy nhiên hãy xem đoạn code dưới đây có an toàn hơn so với đoạn code trên không.

```
// Car2.java

public class Car2 {
  private Engine engine;

  public Engine getEngine() {
    return engine;
  }

  public void setEngine(Engine engine) {
    this.engine = engine;
  }
}
```

Khi nhìn vào những dòng code này, bạn có cảm giác **an toàn hơn** vì chúng có hàm **setter** và **getter** cho thuộc tính **engine**.Tuy nhiên 2 cách viết trong **Car1** và **Car2** chả khác gì nhau.Lý do là chúng ta vẫn có thể sửa đổi thuộc tính **engine** từ **bất kỳ nơi nào** trong code của chúng ta nguyên nhân là **setter** có mức độ truy cập là **public** !!!

Việc lạm dụng setter và getter như trong ví dụ trên khiến code của bạn dài ra và phức tạp 1 cách không cần thiết.

Getter và Setter sẽ tiết lộ chi tiết các class của bạn được triển khai thế nào.



Nhìn vào biểu đồ này ta dễ dàng đoán được tất cả các thuộc tính của class, đoán được class này triển khai như thế nào.Điều này đã **vi phạm tính đóng gói trong lập trình hướng đối tượng**

Việc lạm dụng setter và getter có lẽ sẽ cho ta biết các phương thức có logic và cách thức triển khai như nào!

Getter và Setter có thể gây nguy hiểm cho code của bạn.

Hãy xem đoạn code dưới đây

```
// Debts.java
// Debts: nợ nần, khoản nợ
public class Debts {
  private List<Debt> debts;

  public List<Debt> getDebts() {
    return debts;
  }
}
```

Mục đích của phương thức **getDebts** là lấy ra **danh sách các khoản nợ** của 1 đối tượng nào đó.

Nhìn đoạn code này rất bình thường.Bạn nhớ đến **kiểu dữ liệu nguyên thủy** và **kiểu dữ liệu đối tượng** chứ ?

Do **List** là **kiểu dữ liệu đối tượng**.Vì vậy khi gọi phương thức **getDebts** của đối tượng tạo bởi class **Debts** nó sẽ trả về **địa chỉ tham chiếu** đến vùng nhớ trùng với địa chỉ vùng nhớ mà thuộc tính **debts** đang lưu trữ.Như vậy thay vì chỉ lấy **danh sách các khoản nợ nần**, ta hoàn toàn có thể **gọi phương thức add** được cung cấp bởi List để **thêm khoản nợ mới** 1 cách bất hợp pháp !

```
Debts yourDebts = new Debts();
List<Debt> debts = yourDebts.getDebts();

// Hacker có thể thêm khoản nợ mới vào tài khoản của bạn
debts.add(new Debt(new BigDecimal(1000000)));
```

Giải pháp ở đây là ta sẽ tạo ra 1 **bản sao** của thuộc tính **debts** khi gọi phương thức **getDebts**

Sử dụng setter và getter sao cho hợp lý ?

- Việc thêm các phương thức getter và setter nên là phương án cuối cùng sau khi xem xét các phương án khác
- Đối với các thuộc tính có kiểu dữ liệu đối tượng, chúng ta nên cẩn thận khi dùng getter.Và cách tốt nhất là hạn chế việc sử dụng getter với những thuộc tính kiểu này
- Đừng hỏi thông tin bạn cần để thực hiện công việc.Hãy yêu cầu đối tượng chứa thông tin đó thực hiện công việc đó cho bạn!

Sử dụng setter và getter hợp lý sẽ :

- Kiểm soát tốt hơn các thuộc tính và phương thức của lớp
- Thuộc tính lớp có thể được đặt ở chế **độ chỉ đọc** `get` (nếu bạn chỉ sử dụng phương thức get) hoặc **chỉ ghi** `set` (nếu bạn chỉ sử dụng phương thức set)

- Linh hoạt: lập trình viên có thể thay đổi một phần của mã mà không ảnh hưởng đến các phần khác
- Tăng cường bảo mật dữ liệu

> **Đọc thêm những điều thú vị về getter và setter ở đây!**