You are doing validation in Apex wrong!

Use **Chain of Responsibility** design pattern, but what is that?

Chain of Responsibility is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

First declare Interface with next and validate methods:

```
public interface Validation {
    void setNext(List<String> validations);
    Boolean validate(Account record);
}
```

The next step is to implement the Validation interface in a new Class:

```
public class AccountIndustryValidator implements Validation {
    private static final List<String> SUPPORTED_INDUSTRIES = new List<String>{
        'Electronics', 'Engineering'
    };
    private Validation next;

    public void setNext(List<String> validations) {
        if (!validations.isEmpty()) {
            next = (Validation) Type.forName(validations.remove(0)).newInstance();
            next.setNext(validations);
        }
    }

    public Boolean validate(Account account) {
        return SUPPORTED_INDUSTRIES.contains(account?.Industry) &&
               (next == null ? true : next.validate(account));
    }
}
```

Here we are doing our Validation, in this case checking Account's Industry, and if there is another one we call validate on next Validations.

Finally, we can use new Validators, to check if the Account we got is a valid one:

```
public with sharing class AccountController {
    private static final List<String> VALIDATIONS = new List<String>{
        'AccountIndustryValidator', 'AccountTypeValidator'
    };

    public static Account createAccount(Account account) {
        Validation validation;

        if (!VALIDATIONS.isEmpty()) {
            validation = (Validation) Type.forName(VALIDATIONS.remove(0)).newInstance();
            validation.setNext(VALIDATIONS);
        }

        if (validation.validate(account)) {
            insert account;
        }

        return account;
    }
}
```

Validators are created dynamically, and we have to run only the first one using the validate method. This design is very flexible and can be more configurable, for example, you can use Custom Metadata to store Validator Class names or create more complex validations!