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.

beyond the cloud

First create Validator abstract class:

```
public abstract class Validator {
    protected Validator next;

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

        return this;
    }

    public abstract Boolean isValid(Account record);
}
```

And then extend it with your custom isValid, validation method:

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

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

beyond the cloud

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

```
public with sharing class AccountController {
    public static Account createAccount(Account account) {
        Validator validator = new AccountIndustryValidator()
            .setNext(new List<String>{ 'AccountTypeValidator' });

        if (validator.isValid(account)) {
            insert account;
        }

        return account;
    }
}
```

We create the first Validator manually, then all the logic is handled recursively thanks to implementation in an abstract class. In this implementation we are leveraging Dynamic Apex, it gives a lot of flexibility, you can change implementation to use, for example, Custom Metadata!