PARA QUE SERVE O ENCAPSULAMENTO?

A ideia central é a de – como denota o nome – "encapsular" códigos e métodos. Neste contexto, isso significa **fazer com que o usuário final tenha acesso ao mínimo possível da composição total do produto**.

Assim, para alcançar esta finalidade, o primeiro passo é transformar os atributos do objeto em elementos privados, isto é, que só podem ser acessados pelo próprio objeto, e não por um usuário, por exemplo. Assim, este ao invés de ter o poder de modificar diretamente um atributo do objeto, recorre a uma interface projetada para servir de ponte para o acesso as funcionalidades disponíveis.

Poderíamos citar um controle, que é um objeto que possui atributos diversos que o próprio usuário não tem acesso direto, e sequer conhecimento da existência, pois para a utilização adequada basta que utilize a interface – os botões do controle. São estes que quando clicados chamam métodos do objeto que por sua vez alteram os atributos privados e internos do controle, mudando seu estado, e atendendo as exigências temporais do usuário.

Sendo breve, e para tornar mais claro, podemos afirmar que para aumentar o volume de um dispositivo, como uma televisão, utilizamos o botão adequado do controle, ao invés de abrir o dispositivo e manipular diretamente os sinais elétricos para que altere sua forma de emitir o som. Assim, dizemos que os atributos que o usuário normalmente não tem acesso, por não precisar ter, ou até mesmo por não poder ter, são privados, e aqueles que tem acesso por serem o meio para uso do produto são públicos.

Assim, nesta fase introduzimos também a "**classe abstrata de interface**", que diferentemente de uma classe comum, (i) não origina um objeto em si, não sendo um "molde", como comumente entendida a classe comum, e (ii) possui métodos, mas cuja codificação não é realizada dentro de si mesma, e sim implementada em uma classe comum que origina um objeto.

Criamos, por exemplo, a classe abstrata "Interface_celular" e seus métodos, apenas listados um a um, sem tarefas contidas em si. Após isso, a implementamos por meio da função "implements", no arquivo da classe comum "Celular", que é, como sabemos, um molde para criação de objetos. Neste, irão ser definidos os atributos privados da classe "Celular", os métodos de getter e setter para modificação destes, e então os métodos descritos no arquivo "Interface_celular", com tarefas descritas. E é melhor que isto seja feito nesta ordem, no arquivo:

DECLARAÇÃO DOS ATRIBUTOS

CRIAÇÃO DOS MÉTODOS GETTERS E SETTER

CODIFICAÇÃO DAS TAREFAS DOS MÉTODOS DA INTERFACE

E que tarefas serão? Estes métodos da "interface" irão chamar os métodos getter e setter, que por sua vez irão modificar os atributos do objeto. Por exemplo, o método público

“ligar()”, de interface, irá conter a tarefa “$this->setLigado(true)”, que irá enviar o valor true como parâmetro para o método setter “setLigado($var)”, que por sua vez irá atribuir o valor recebido ao atributo “private $ligado” do objeto.

Desta forma se torna claro que a relação entre estas camadas do objeto são análogas ao uso concreto de um objeto como um controle; pois (i) clicamos no botão ligar da interface, (ii) esta ação dispara um sinal – setter- no circuito do controle, que (iii) contata o dispositivo televisão para alterar o atributo de volume.