Encapsulamento

Fortemente relacionado com o Conceito Classe e Objeto, temos o encapsulamento ou encapsulação que significa esconder informação. Objetiva-se com o encapsulamento proteger o estado e o comportamento interno do objeto, disponibilizando somente uma camada de interface com o ambiente externo para que seja efetuada a comunicação com o objeto.

O reflexo do encapsulamento na Orientação a Objeto é que todo o objeto na sua estrutura elementar possui uma membrana de proteção ao longo do seu núcleo. O acesso ao estado e comportamento só é permitido através da passagem pela camada de comunicação representada pela membrana. Em programação, somente a assinatura do método, ou seja, a chamada da função está publicada na interface. O conjunto da assinatura de todos os métodos também é conhecida como API (Application Programming Interface).

Para um desenvolvedor utilizar objetos é necessário que ele conheça somente API dos mesmos. A complexidade de como um Objeto foi implementado está encapsulada, não sendo relevante neste processo. Pode-se ilustrar esta situação através da seguinte análise: quando se dirige um carro não é necessário compreender como o motor funciona com todas as suas ‘milhares’ de peças. O motorista abstrai a complexidade interna do motor e fixa-se na interface disponibilizada por quem construiu o carro, portanto, as interfaces seriam os pedais de aceleração, freio e embreagem, painel com medidor de velocidade etc.

A forma de garantir o encapsulamento de atributos e métodos é determinar a sua visibilidade privada através dos modificadores de acesso disponíveis nas linguagens de programação (private ou protected). Por exemplo, normalmente um atributo na linguagem java é declarado private String nome, sendo private uma palavra-chave indicando que a visibilidade do atributo nome só é permitida para métodos da própria classe.

A definição básica sobre a abstração é supressão de detalhe, visando controlar a complexidade pela ênfase em características essenciais. Na Orientação Objeto, pode-se aplicar este conceito no processo de especificação, construção e uso de classes.

Observa-se que os objetos identificados no mundo real normalmente são muito complexos. Por exemplo, imagine o que seria modelar uma classe Pessoa com todos os atributos e comportamentos!

Para a redução da complexidade da Classe Pessoa, é necessário delimitar o escopo com base no domínio do sistema que esta Pessoa será modelada, considerando somente os aspectos relevantes. Por exemplo, para um sistema acadêmico, uma Pessoa pode representar um aluno,

sendo nesse escopo somente necessário os atributos nome, matrícula e idade. As demais características que não são essenciais nesse contexto serão abstraídas. Como os detalhes ficam internos e escondidos no objeto Pessoa, podese subentender que este objeto possui quaisquer características de uma Pessoa do mundo real. A abstração também se aplica na construção de uma classe. Uma primeira etapa é definir toda a interface da classe, ou seja, como os componentes externos irão interagir com a classe e qual o comportamento esperado pela mesma. O objetivo nesta etapa é somente de definir a interface (protocolo de comunicação), focando no que a classe deve fazer e não como a interface será implementada. Portanto, abstrai-se a implementação interna e concentra-se na especificação da interface que representa o protocolo de comunicação com o meio externo. Um bom critério de programação para a redução do acoplamento é programar para uma interface e não para uma implementação. Com a interface bem definida, pode-se abstrair o mundo externo e focar-se em como realizar o que foi especificado na interface.

A meta de um desenvolvedor no projeto e implementação de um software deve ser sempre obter o máximo de redução de acoplamento e complexidade. É dito redução, pois um software sempre terá algum nível de acoplamento, visto os objetos estarem relacionados e trocarem mensagens entre si. Um projeto de software Orientado a Objeto que respeita a propriedade do encapsulamento, mesmo não usando nenhuma outra técnica, já está em um bom caminho para alcançar a redução de acoplamento e complexidade.

Portanto, uma vantagem básica da encapsulação é facilitar mudanças isolando pedaços uns dos outros (reduzindo o acoplamento). Para melhor entendimento de como o encapsulamento pode beneficiar o redução do acoplamento, ilustra-se com um exemplo prático. No contexto do cálculo da autonomia de um carro, supondo que exista uma regra de negócio mapeada pela formula R(1) = (consumo * qtdLitros). Neste exemplo, têm-se três objetos a, b e c que necessitam da informação da autonomia. Para isso, esses três objetos acessam um quarto objeto do tipo Carro, obtendo seus dados de consumo e qtdLitros.

O acoplamento entre os objetos pode fazer com que uma mudança tenha repercussão em vários pontos do software, dificultando o processo de manutenção. Para o exemplo anterior, suponha, agora, que haja uma alteração na regra de negócio referente à autonomia, e a formula seja mudada para R(1) = (consumo * qtdLitos * percentualPerda). O resultado desta simples mudança tem impacto significativo; é uma alteração nos objetos a,b, c.. A forma correta para esta situação é encapsular o algoritmo referente à regra de negócio R(1) no objeto Carro e disponibilizar, na sua interface, um método double getAutonomiaKm() . Assim, caso haja alguma mudança na fórmula da autonomia do Carro, esta fica restrita ao objeto Carro.

Outro critério que o desenvolvedor deve buscar ao longo do projeto de um software é a redução da complexidade do mesmo.

A técnica de modularização, amplamente utilizada no paradigma funcional, estabelece que, para problemas complexos, deve-se dividi-los em partes menores, criando unidades funcionais menores de mais fácil compreensão e desenvolvimento.

Caso seja aplicável devido ao tamanho de um projeto, deve decompor o sistema em subsistemas menores para reduzir a complexidade da situação original. Ao se quebrar o problema em partes menores e mais simples, é possível identificar e estabelecer as camadas e os módulos do sistema, e suas interdependências.

A divisão em camadas é uma estrutura desejável, pois reduz sensivelmente o acoplamento dos subsistemas, visto que, uma camada é um subsistema que interage apenas com o subsistema imediatamente inferior ou superior.

Os módulos (chamados de pacotes, em Java) são unidades menores que compõem os subsistemas e agrupam as unidades funcionais – as classes. Um critério que deve ser adotado para encapsular classes em um módulo é que as classes que pertencem a um mesmo módulo devem ter um maior nível de acoplamento entre si, ao contrário das classes pertencentes a módulos distintos, que possuem um baixo nível de acoplamento.

Bons Estudos e Realize os exercícios para reter o que foi abordado neste módulo