Programação Orientada a Objetos

Encapsulamento

Cristiano Lehrer, M.Sc.

ybadoo

Introdução (1/2)

- Os **três pilares** da programação orientada a objetos são:
  - Encapsulamento
  - Herança
  - Polimorfismo
- **Encapsulamento** é a característica da orientação a objetos de ocultar partes independentes da implementação.
  - O encapsulamento permite que se construa partes ocultas da implementação do software, que atinjam uma funcionalidade e ocultam os detalhes de implementação do mundo exterior.

ybadoo

Introdução (2/2)

- O verdadeiro encapsulamento é imposto em nível de linguagem, através de construções internas da linguagem.
- Qualquer outra forma de encapsulamento é simplesmente um acordo de cavalheiros, que é facilmente malogrado.
- Os programadores o contornarão porque podem fazer isso!

ybadoo

Interface (1/3)

- Uma **interface** lista os serviços fornecidos por um componente.
  - A interface é um contrato com o mundo exterior, que define exatamente o que uma entidade externa pode fazer com o objeto.
  - Uma interface é o painel de controle do objeto.
- A **implementação** define como um componente realmente fornece um serviço.
  - A implementação define os detalhes internos do componente.

ybadoo

Interface (2/3)

```
public class Log {

  public void info(String message) {
    print("INFO", message);
  }

  public void warning(String message) {
    print("WARNING", message);
  }

  public void error(String message) {
    print("ERROR", message);
  }

  public void fatal(String message) {
    print("FATAL", message);
    System.exit(0);
  }

  private void print(String severity, String message) {
    System.out.println(severity + " : " + message);
  }
}
```

Interface (3/3)

- Observações:
  - Uma interface é importante, pois ela diz o que se pode fazer com o componente.
  - O mais interessante é que uma interface não informa como o componente fará seu trabalho.
  - Em vez disso, a interface oculta a implementação do mundo exterior.
  - Isso libera o componente para alterações na sua implementação a qualquer momento.
  - As mudanças na implementação não mudam o código que usa a classe, desde que a interface permaneça inalterada.
  - As alterações na interface necessitarão de mudanças no código que exerce essa interface.

ybadoo

Visibilidade

- Público
  - Garante o acesso a todos os objetos.
  - `public void add(int a, int b)`
- Protegido
  - Garante o acesso à instancia, ou seja, para aquele objeto, e para todas as subclasses.
  - `protected void add(int a, int b)`
- Privado
  - Garante o acesso apenas para a instância, ou seja, para aquele objeto.
  - `private void add(int a, int b)`

ybadoo

Ocultação da Implementação (1/4)

- O código **fracamente acoplado** é independente da implementação de outros componentes.
- O código **fortemente acoplado** é fortemente vinculado à implementação de outros componentes.
- **Código dependente** é dependente da existência de determinado tipo.
  - O código dependente é inevitável.
  - Entretanto, existem graus para a dependência aceitável e para a super-dependência.

ybadoo

Ocultação da Implementação (2/4)

- O encapsulamento e a ocultação da implementação não são mágica.
- Se for preciso alterar uma interface, será preciso atualizar o código que é dependente da interface antiga.
- Ocultando os detalhes e escrevendo software para uma interface, se cria software que é fracamente acoplado.
- O software fortemente acoplado anula o objetivo do encapsulamento:
  - Criar objetos independentes e reutilizáveis.

ybadoo

Ocultação da Implementação (3/4)

- A ocultação da implementação tem seus inconvenientes:
  - Existem ocasiões em que se precisa saber um pouco mais do que a interface pode informar.
  - Ao definir uma interface, é importante não apenas fornecer uma interface, mas também documentar detalhes específicos sobre a implementação.
  - Entretanto, assim como em qualquer outra parte da interface pública, uma vez que se declara um comportamento, não pode alterá-lo.

ybadoo

Ocultação da Implementação (4/4)

- Como se obtém uma ocultação de implementação eficaz e código fracamente acoplado?
  - Só permita acesso as informações através de uma interface baseada em método.
    - Tal interface garante que não se exponha informações sobre a implementação.
  - Não forneça acesso involuntário a estruturas de dados internas, retornando ponteiros ou referências acidentalmente.
    - Após alguém obter uma referência, a pessoa pode fazer tudo com ela.
  - Nunca faça suposições sobre os outros tipos utilizados.
    - A não ser que um comportamento apareça na interface ou na documentação, não conte com ele.
  - Cuidado enquanto escrever dois tipos intimamente relacionados.
    - Não programe acidentalmente em suposições e dependências.

ybadoo

Divisão de Responsabilidade (1/2)

- Cada objeto deve executar uma função, sua responsabilidade, e executá-la bem.
- A ocultação da implementação e a responsabilidade andam lado a lado.
  - Sem ocultação da implementação, a responsabilidade pode faltar em um projeto.
  - É de responsabilidade do objeto saber como fazer seu trabalho.
  - Se deixar a implementação aberta para o mundo exterior, um usuário poderá começar a atuar diretamente na implementação, duplicando assim a responsabilidade.

ybadoo

Divisão de Responsabilidade (2/2)

- Quando dois objetos começam a fazer a mesma tarefa, sabe-se que não tem uma divisão de responsabilidade correta.
  - Quando se observa a existência de lógica redundante, precisará refazer o código.
  - Refazer o trabalho é uma parte esperada do ciclo de desenvolvimento orientado a objetos.
    - À medida que os projetos amadurecem, encontra-se muitas oportunidades de melhorias.

ybadoo

Variáveis/Métodos de Classe

- Variáveis de classe são variáveis que pertencem à classe e não a uma instância específica.
  - As variáveis de classe são compartilhadas entre todas as instâncias da classe.
- Métodos de classe são métodos que pertencem à classe e não a uma instância específica.
  - A operação executada pelo método não é dependente do estado de qualquer instância.

ybadoo