Programação I

Abstração e Encapsulamento

Samuel da Silva Feitosa

Aula 7

Introdução

- Nesta aula vamos estudar o segundo pilar da orientação a objetos.
  - Conceito de **encapsulamento** e as motivações para seu uso.
  - Diversas funcionalidades do Java para permitir o desenvolvimento de código que usa este conceito.
  - Uso de métodos **construtores** para facilitar a criação de objetos.

Relembrando: Primeiro Pilar da Orientação a Objetos

Abstração é a habilidade de se concentrar nos aspectos **essenciais** de um contexto qualquer, **ignorando** características menos importantes ou acidentais.

- Em modelagem orientada a objetos, uma **classe** é uma **abstração** de entidades existentes no domínio do sistema de software.

Encapsulamento

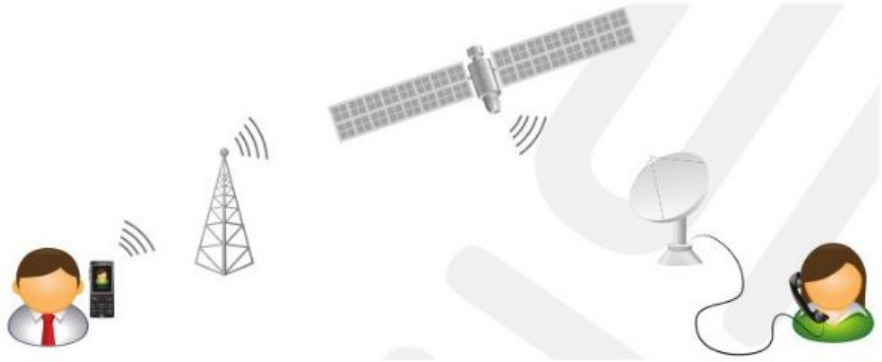
- Segundo pilar da orientação a objetos.
- Estabelece que:
  - Os diferentes componentes de um sistema de software **não devem revelar detalhes** de suas respectivas implementações.
  - Oferece ao programador liberdade na implementação dos detalhes do sistema.
- A única restrição ao programador é **manter a interface** percebida pelos usuários.
  - Interface é o conjunto de **atributos** e **métodos** públicos de uma classe.

Por quê encapsular?

- Encapsular significa esconder a implementação dos objetos.
- Favorece a manutenção e desenvolvimento.
  - **Manutenção:** quando o funcionamento de um objeto deve ser modificado, basta modificar a classe do mesmo.
  - **Desenvolvimento:** é possível determinar precisamente as responsabilidades de cada classe da aplicação.

Exemplo - Celular

- Os botões, a tela e os menus formam a **interface de uso**.
- Os **dispositivos internos** e os **processos** que transformam o som captado em ondas que podem ser transmitidas constituem a **implementação**.



Exemplo - Carro

- A **interface de uso** de um carro é composta pelos dispositivos que permitem que o motorista **conduza** o veículo (volante, pedais, alavanca de câmbio, etc.).
- A **implementação** é composta pelos dispositivos **internos** (motor, caixa de câmbio, radiador, etc.) e pelos **processos** realizados internamente.

Exemplo - Máquina de vendas

- Todos já devem ter utilizado alguma vez uma máquina de refrigerantes, de salgadinhos, de café, etc.
  - Entradas para moedas ou cédulas.
  - Botões para escolher o produto.
  - Saída do produto.
  - Saída para troco.
- Sistemas deste tipo devem **proteger** o acesso a suas **funcionalidades internas**, para que seu **funcionamento externo** não seja prejudicado.

Modificadores de Visibilidade (1)

- Permitem que o programador restrinja o uso de certos elementos da classe.
  - Permite **encapsular** as informações.
- Java possui três especificadores de acesso:
  - **public:** pode ser usado livremente pelas instâncias da classe.
  - **protected:** só pode ser usado na implementação de subclasses.
  - **private:** não pode ser usado fora da implementação da própria classe.

Modificadores de Visibilidade (2)

- O encapsulamento provido pelos modificadores de visibilidade possibilita:
  - Obter **código mais claro**, pois os membros com funções reais são diferenciados dos auxiliares.
  - Ocultar **detalhes de implementação**.
  - **Simplificar as interfaces** das classes.
  - Proporcionar facilidades para **extensão** (criação de subclasses).
  - **Facilitar as modificações**, pois, se a interface permanece inalterada, as alterações tornam-se transparentes.

Exemplo - Modificadores

- Exemplo de classe que representa um controlador de uma máquina de Vidro Elétrico de um carro.

```
public class VidroEletrico {
    private int posicao;
    public boolean aberto;

    public void baixarVidro() {
        if (posicao > 0) {
            posicao -= 2;
        } else {
            System.out.println("Vidro já está todo aberto!");
        }

        aberto = true;
    }

    public void subirVidro() {
        if (posicao >= 10) {
            aberto = false;
            System.out.println("Vidro já está todo fechado!");
        } else {
            posicao += 2;
        }
    }
}
```

Métodos de Acesso - Get e Set

- Na linguagem Java, há uma convenção de nomenclatura para os métodos que têm como finalidade acessar ou alterar propriedades.
  - **get:** permite a **consulta** das propriedades.
  - **set:** permite **alterar** as propriedades de um objeto.
- É muito conveniente seguir essa convenção, pois desenvolvedores Java estão acostumados com essa regras, e o funcionamento de muitas bibliotecas depende deste padrão.

Exemplo - Get e Set

- Usando métodos get e set para controlar a posição do vidro do carro.

```
public class VidroEletrico {
    private int posicao;
    public boolean aberto;

    public void baixarVidro() {...9 lines }

    public void subirVidro() {...8 lines }

    public int getPosicao() {
        return posicao;
    }

    public void setPosicao(int pos) {
        if (posicao >= 0 && posicao <= 10) {
            this.posicao = pos;
        }
        else {
            System.out.println("Posição fora dos limites permitidos");
        }
    }
}
```

Construtores (1)

- São métodos especiais destinados inicialização e ao preparo de novos objetos.
  - Assim como métodos comuns, os construtores podem receber parâmetros.
- Só podem ser acionados usando **new**.
  - Construtores devem possuir o **mesmo nome** da classe e **não possuem** tipo de retorno.
  - O programador não é obrigado a incluir construtores. Neste caso, o compilador inclui o construtor *default*.

Construtores (2)

- Normalmente os construtores são usados para definir valores iniciais para os atributos do objeto.
  - Isso garante um estado inicial consistente ou simplifica o uso das instâncias.
- Construtores são geralmente declarados como **públicos**.
  - Em casos especiais, eles podem ser definidos como **protegidos** ou **privados**.

Exemplo - Construtores

- Exemplo de construtores para a classe VidroEletrico.
  - Note que existem dois construtores com o mesmo nome. Isto se chama **sobrecarga** de construtores.

```
public class VidroEletrico {
    private int posicao;
    public boolean aberto;

    public VidroEletrico() {
        this.posicao = 10;
        this.aberto = false;
    }

    public VidroEletrico(int posicao, boolean aberto) {
        this.posicao = posicao;
        this.aberto = aberto;
    }

    public void baixarVidro() {...9 lines }

    public void subirVidro() {...8 lines }

    public int getPosicao() {...3 lines }

    public void setPosicao(int pos) {...8 lines }
}
```

Considerações Finais

- Nesta aula foi possível estudar o segundo pilar da orientação a objetos, o **encapsulamento**.
  - Vimos que com isso é possível isolar os componentes desenvolvidos, o que traz diversos benefícios para um projeto.
  - O uso de **modificadores de visibilidade** e métodos **getters** e **setters** são essenciais para implementar este conceito.
- Construtores são **métodos especiais** úteis para a **inicialização** rápida de atributos de objetos.

Exercício - OOP1

- Implementar uma classe Java que represente **outro componente** de um Carro, similar ao que foi feito com a máquina de vidro elétrico.
  - Defina pelo menos **três atributos**, o **construtor**, os **métodos get** e **set**, e alguma outra operação que este componente executa em um carro.

Dica: Criem uma conta no GitHub com um repositório para armazenar os códigos da disciplina. Estes códigos podem ser úteis em estudos futuros.