Análise e Desen. de Sistemas

POO

Programação Orientada a Objetos I

Prof. Letícia Pieper

60 h

UNIFASIPE
CENTRO UNIVERSITÁRIO

Encapsulamento

Um dos pilares da orientação a objetos, o encapsulamento trata da segurança das propriedades a fim de escondê-las e protegê-las da aplicação.

A aplicação pede o acesso à propriedade e passa a acessá-la através de métodos e funções.

Define níveis de visibilidade

public

Indica que a classe, atributo ou objeto é visível em qualquer pacote.

private

Indica que o método é visível somente dentro da classe onde foi declarado.

default

Indica que o método é visível somente dentro do pacote que foi declarado.

protected

indica que o método é visível dentro do pacote ou em classes filhas.

#Fundamentos

1. Para obter o resultado de um uma propriedade usa-se o conceito de GET.

2. Para atribuir um valor à propriedade usa-se o conceito de SET.

#Fundamentos

```
public class Carro {
    String marca;
    String modelo;
    int ano;

    private int hodometro;
    int kmRodado(){
        return hodometro;
    }

    void andar(int km){
        this.hodometro = this.hodometro + km;
    }
}
```

Também servem para proteger métodos que não devem ser acessados diretamente pelo usuário.

Exemplo:

Controle Televisão.

#Fundamentos

```
public class Controle {
    private boolean ligado;

    private void ligar(){
        this.ligado = true;
        System.out.println("Você ligou a TV.");
    }

    private void desligar(){
        this.ligado = false;
        System.out.println("Você desligou a TV.");
    }

    void botaoPower(){
        if (this.ligado) {
            this.desligar();
        } else {
            this.ligar();
        }
    }
 }
```

```
public enum Naipe {
    OURO("Vermelho"),
    PAUS("Preto"),
    COPAS("Vermelho"),
    ESPADAS("Preto");

    Naipe(String cor){
        this.cor = cor;
    }

    private String cor;
    public String getCor(){
        return this.cor;
    }
 }
```

Enum no Java

São tipos de campos que consistem em um conjunto fixo de constantes (static final), sendo como uma lista de valores pré-definidos. Na linguagem de programação Java, pode ser definido um tipo de enumeração usando a palavra chave ***enum***.

```
public class Livro {
    private String titulo;
    private String autor;

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
}
```

Note que os dois atributos da classe (titulo e autor) são do tipo **private**, que permite apenas o acesso destas informações a partir da própria classe, logo para que seja possível ler ou alterar essas informações criamos métodos ditos **métodos assessores** ou então **getters** e **setters**.

A princípio parece ser algo sem muita utilidade, mas desta forma podemos criar atributos os quais podemos apenas ler informações ou ainda gerar tratamentos específicos sempre que outra classe solicita a alteração de um atributo de nossa classe.

Encapsule aquilo que pode ser alterado com frequência na sua aplicação, por exemplo:

```
public class Professor {
    private int registro;
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getRegistro() {
        return registro;
    }

    public void setRegistro(int registro) {
        this.registro = registro;
    }
}
```

Encapsule aquilo que pode ser alterado com frequência na sua aplicação, por exemplo:

```
public class Aluno {
    private int matricula;
    private String nome;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getMatricula() {
        return matricula;
    }

    public void setMatricula(int matricula) {
        this.matricula = matricula;
    }
}
```

Note que os atributos das classes **Professor** e **Aluno** possuem a visibilidade **private**, dessa forma eles só podem ser acessados pelas suas classes, também criamos métodos **set** (métodos utilizados para alterar o valor de uma propriedade) e **get** (método utilizado para obter o valor de uma propriedade) para cada atributo.

Dado a classe Professor e Aluno, queremos consultar os alunos e professores pelo nome:

Implementar as Classes identificadas abaixo, aplicando o princípio do Encapsulamento.

#Fundamentos

Crie uma classe Produto para representar um produto do mundo real. Sua classe deverá conter os seguintes atributos e métodos:

1) Um campo de dados privado do tipo String chamado nome, que representará o nome do produto.

2) Um campo de dados privado do tipo double chamado precoCusto, que guardará o preço de custo do produto.

3) Um campo de dados privado do tipo double chamado precoVenda, que guardará o preço de venda do produto.

4) Um campo de dados privado do tipo double chamado margemLucro, que guardará a margem de lucro do produto.

5) Métodos públicos get() e set() para os atributos acima. Modifique o método setPrecoVenda() para que o preço de venda não seja inferior ao preço de compra. Caso isso aconteça, exiba uma mensagem alertando o usuário.

6) Crie um método chamado calcularMargemLucro() que calculará a margem de lucro do produto.

7) Crie um método chamado getMargemLucroPorcentagem() que retornará a margem de lucro como percentual.

Para finalizar, no método main() da classe de teste, crie um novo objeto da classe Produto, peça para o usuário informar os preços de custo e de venda e exiba a margem de lucro em moeda e em percentual.

Sua saída deverá ser algo parecido com o mostrado na imagem ao lado:

```
Informe o preço de custo: 120
Informe o preço de venda: 195
Preço de custo: 120.0
Preço de Venda: 195.0
Margem de Lucro: 75.0
Margem de Lucro Percentual (%): 62.5
```

Adicione o modificador de visibilidade (private, se necessário) para cada atributo e método da classe Conta. Tente criar uma Conta no main e modificar ou ler um de seus atributos privados. O que acontece?

```
public class Conta {
    private String titular;
    private int numero;
    private String agencia;
    private double saldo;
    private Data dataDeAbertura;

    public void saca(double valor) {...}

    public void deposita(double valor) {...}

    public double calculaRendimento() {...}

    public String recuperaDadosParaImpressao() {...}
}
```

Crie apenas os getters e setters necessários da sua classe Conta. Pense sempre se é preciso criar cada um deles. Por exemplo:

```
class Conta {
    private String titular;

    // ...

    public String getTitular() {
        return this.titular;
    }

    public void setTitular(String titular) {
        this.titular = titular;
    }
}
```

Crie apenas os getters e setters necessários da sua classe Conta. Pense sempre se é preciso criar cada um deles. Por exemplo:

```
class Conta {
    private String titular;

    // ...

    public String getTitular() {
        return this.titular;
    }

    public void setTitular(String titular) {
        this.titular = titular;
    }
}
```

Não copie e cole! Aproveite para praticar sintaxe.

Repare que o método *calculaRendimento* parece também um getter. Aliás, seria comum alguém nomeá-lo de *getRendimento*. Getters não precisam apenas retornar atributos, eles podem trabalhar com esses dados.

Altere suas classes que acessam e modificam atributos de uma `Conta` para utilizar os getters e setters recém-criados.

Por exemplo, onde você encontra:

```
  c.titular = "Batman";
   System.out.println(c.titular);
```

passa para:

```
  c.setTitular("Batman");
   System.out.println(c.getTitular());
```

Crie uma classe denominada Elevador para armazenar as informações de um elevador dentro de um prédio.

A classe deve armazenar o andar atual (térreo = 0), total de andares no prédio, excluindo o térreo, capacidade

do elevador, e quantas pessoas estão presentes nele. A classe deve também disponibilizar os seguintes métodos:

- **Inicializar: que deve receber como parˆametros a capacidade do elevador e o total de andares no prédio**

(os elevadores sempre começam no térreo e vazio);

- **Entrar: para acrescentar uma pessoa no elevador (se deve acrescentar se ainda houver espaço);**
- **Sair: para remover uma pessoa do elevador (só deve remover se houver alguém dentro dele);**
- **Subir: para subir um andar (n˜ao deve subir se já estiver no último andar);**
- **Descer: para descer um andar (não deve descer se já estiver no térreo);**

Encapsule todas as propriedades usando os métodos set e get.

https://www.arquivodecodigos.com.br/dicas/3439

http://www.ruirossi.pro.br/livros/li-rui-pcj-cap13.pdf

http://www.universidadejava.com.br/java/java-encapsulamento/