FADERGS
CENTRO UNIVERSITÁRIO

Programação Orientada a Objetos
Visibilidade

Prof. Flavio Treib

2022/01

FADERGS
CENTRO UNIVERSITÁRIO

Visibilidade

- Indica o nível de acesso aos componentes internos de uma classe (atributos e métodos)
- Podem ser:
  - Public
  - Private
  - Protected

Visibilidade

- + Public
  - Quem tem acesso à classe tem acesso também a qualquer membro com visibilidade public
- - Private
  - somente os objetos da classe detentora do atributo ou método poderão enxergá-lo ou utilizá-lo;
  - O objetivo é ter o controle centralizado dos dados do objeto, facilitando a manutenção e detecção de erros.
- # Protected
  - Além dos objetos da classe detentora do atributo ou método também os objetos de suas subclasses poderão ter acesso ao mesmo.

Exemplos

- Objeto Telefone
  - telefone publico: orelhão, qualquer um pode usar
  - telefone privado: celular, só eu uso
  - telefone protegido: telefone de casa: só quem é da família: minha mãe e todos os filhos dela

- Todo método tem visibilidade pública para pacote, por padrão

Programação Orientada a Objetos
Encapsulamento

Material inicialmente elaborado pelo Prof. Flávio Treib e adaptado pelo Prof. Adalto Selau Sparremberger

Encapsulamento

- Técnica que faz com que detalhes internos do funcionamento dos métodos de uma classe permaneçam **ocultos** para os objetos

- Encapsular seria o mesmo que esconder todos os membros de uma classe, além de esconder como funcionam as rotinas (no caso métodos) do nosso sistema. Seria uma espécie de proteção

- Encapsular é fundamental para que seu sistema seja suscetível a mudanças: não precisaremos mudar uma regra de negócio em vários lugares, mas sim em apenas um único lugar, já que essa regra está encapsulada

Encapsulamento

- Para protegermos os atributos e métodos, por tanto, podemos mudar seus modificadores. O principal deles é o modificador *private* que faz com que ninguém consiga modificar, nem mesmo ler, o atributo em questão.

- No exemplo a seguir, temos uma classe chamada Funcionário com um atributo do tipo Double chamado salário:

```
class Funcionario{
        double salario;
}
```

Encapsulamento

- No exemplo, o atributo salario pode ser acessado ou modificado por código escrito em qualquer classe que esteja no mesmo diretório que a classe Funcionario. Portanto, o controle desse atributo é descentralizado.

- Como fazer para proteger o acesso a este atributo para que ele não seja alterado por qualquer classe?

- Podemos fazê-lo incluindo o modificador Private no atributo.

Encapsulamento

```
class Funcionario {
  private double salario ;

  void aumentaSalario ( double aumento ) {
  // lógica para aumentar o salário
  }
}
```

Métodos Privados

- O papel de alguns métodos pode ser o de auxiliar outros métodos da mesma classe. E muitas vezes, não é correto chamar esses métodos auxiliares de fora da sua classe diretamente.

- No próximo exemplo, temos o método descontaTarifa() como um método auxiliar dos métodos deposita() e saca(). Além disso, ele não deve ser chamado diretamente, pois a tarifa só deve ser descontada quando ocorre um depósito ou um saque

Métodos Privados

```
class Conta {
 private double saldo ;

 void deposita ( double valor ) {
   this.saldo += valor ;
   this.descontaTarifa ();
 }

 void saca ( double valor ) {
   this.saldo -= valor ;
   this.descontaTarifa ();
 }
// Método privado para que não seja alterado fora da classe
 private void descontaTarifa () {
   this.saldo -= 0.1;
 }
}
```

Métodos Públicos

- Para que um método seja visível em outras classes, é necessário que ele seja do tipo Public. Do contrário, seu modificador será o padrão (default), que permite que ele seja visível apenas dentro do próprio pacote de criação.

- Os pacotes em Java são os responsáveis pela organização do projeto e pela proteção de parte do código em restrições específicas

Métodos Públicos

```
class Conta {
 private double saldo ;

// Método público
 public void deposita ( double valor ) {
   this.saldo += valor ;
   this.descontaTarifa ();
 }
// Método público
 public void saca ( double valor ) {
   this.saldo -= valor ;
   this.descontaTarifa ();
 }
// Método privado
 private void descontaTarifa () {
   this.saldo -= 0.1;
 }
}
```

Porque encapsular?

- A manutenção é favorecida pois, uma vez aplicado o encapsulamento, quando o funcionamento de um objeto deve ser alterado, em geral, basta modificar a classe do mesmo.

- O desenvolvimento é favorecido pois, uma vez aplicado o encapsulamento, conseguimos determinar precisamente as responsabilidades de cada classe da aplicação.

Métodos acessores e modificadores

- Geralmente, os campos de dados privados são de natureza técnica e interessam apenas ao criador das operações.
- Assim, o acesso aos atributos deve ser permitido pela implementação por meio de três itens:
  - Um campo de dados privado
  - Um método de leitura (acessador)
  - Um método de alteração (modificador)
- Por convenção, acessores e modificadores são chamados Getter and Setter

Métodos Acessores e Modificadores

- Vantagens: Implementação interna pode ser modificada sem afetar nenhum código fora da própria classe.

- Os métodos “modificadores” podem fazer testes contra erros

Métodos Acessores e Modificadores

- **Set**
  - Nomeamos um método modificador com **set** toda vez que este método for modificar algum campo ou atributo de uma classe, ou seja, se não criarmos um método modificador **set** para algum atributo, isso quer dizer que este atributo não deve ser modificado.

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

Métodos Acessores e Modificadores

- **Get**
  - Nomeamos um método acessor com get toda vez que este método for verificar algum campo ou atributo de uma classe.
  - Como este método irá verificar um valor, ele sempre terá um retorno como String, int, float, etc. Mas não terá nenhum argumento.

```
public String getNome(){
    return nome;
}
```

Métodos
Acessores e Modificadores

```
public class Ponto {
        private double x;
        private double y;

        public Ponto(double x, double y) {
                this.x = x;
                this.y = y;
        }

        public double getX() { return x; }

        public double getY() { return y; }

        public void setX(double x) { this.x = x; }

        public void setY(double y) { this.y = y; }

}
```

Exercício

- *Crie um projeto banco no netbeans*
- *Crie a classe Conta, com os seguintes atributos:*
  - *numero*
  - *tipo (conta corrente/conta poupança)*
  - *dono*
  - *saldo*
  - *status (aberta/fechada)*
- *Crie os métodos assessores e modificadores*
- *Crie os seguintes métodos:*
  - *Abrir Conta*
  - *Fechar conta*
  - *depositar*
  - *sacar*
  - *pagar mensalidade*

Exercício

- *Regras:*
  - *Toda a nova conta, ao ser criada, deve receber status fechada e saldo zero.*
  - *Ao abrir uma conta, se esta for do tipo CC (Conta Corrrente) deverá receber R$ 50,00 como saldo inicial. Se for do tipo CP (Conta Poupança) deverá receber R$ 150,00 de saldo inicial.*
  - *Ao fechar a conta, verificar se o saldo é ZERO. Se saldo for positivo, será necessário sacar o valor, se for negativo será necessário depositar o valor.*
  - *Para realizar um depósito, verificar se a conta está aberta. Se aberta, incrementar o saldo, se fechada avisar o usuário.*
  - *Para sacar, informar o valor do saque e verificar se a conta está ativa e com saldo suficiente. Se sim, realizar o saque e atualizar o saldo, caso contrário, avisar o cliente.*
  - *O custo da mensalidade é de R$ 12,00 para conta corrente e R$ 25,00 para conta poupança. Essa cobrança só poderá ser realizada se o saldo for suficiente.*

Atributos Estáticos

- Atributo que pertence à classe e não ao objeto.
- Representado com o comando *static* na declaração do atributo
- Ex.:

```
public class Conta {
        private static int numeroConta;
        .
        .
        .
        .
```

Métodos Estáticos

- Método que pertence à classe e não ao objeto.
- Não é necessário instanciar um objeto para utilizar o método estático
- Representado com o comando *static* na declaração do método
- Ex.:

```
public static int ProximaConta()
   {
      return Conta.numeroConta++;
   }
```

Métodos Estáticos

- Para chamar o método:

Conta.ProximaConta();

FADERGS

CENTRO UNIVERSITÁRIO