



Java Standard Edition (JSE)

Capítulo 05. Encapsulamento, Modificadores de acesso e atributos de classe



Esp. Márcio Palheta

MSN: marcio.palheta@hotmail.com



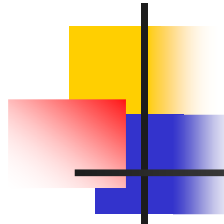
Agenda

- Revisão da aula anterior;
- Motivação – Organização;
- Controlando o acesso;
- Encapsulamento;
- Getters e Setters;
- Construtores;
- Atributos de classe;
- Exercícios de fixação



Revisão

- Laços de repetição for e while;
- Paradigma das Orientação a Objetos
- Classe, objeto, atributos e métodos;
- Acesso a atributos de uma classe;
- Relacionamento entre classes;
- Arrays dinâmicos;
- Dúvidas ?



Motivação – Organização

- A organização pessoal ajuda no desenvolvimento das tarefas do cotidiano;
- Planeje sua formação;
- Identifique os cursos necessários e o custo do investimento;
- Estabeleça metas a médio e longo prazo;
- Vídeos:
 - 05.01 Organizacao;
 - 05.02 Tudo a seu tempo



Novos recursos a aprender

- Implementar controle de acesso aos seus métodos, atributos e construtores, através dos modificadores `private` e `public`;
- Escrever métodos de acesso a atributos do tipo `getters` e `setters`;
- Escrever construtores para suas classes;
- Utilizar variáveis e métodos estáticos.



Controle de acesso

- O método sacar tem como objetivo atualizar o saldo da conta, evitando que seja sacado um valor maior que o saldo;

```
public class ContaBancaria {  
    String numeroConta;  
    double saldo;  
    double limite;  
    Cliente titular;  
    public boolean sacar(double valor) {..  
    public void depositar(double valor) {..  
    public boolean transferir(ContaBancaria  
    public double getSaldo() { ..  
}
```



Controle de acesso

- Mas existe uma forma de “burlar” o método sacar() e alterar o valor do saldo de forma direta:
 - `ContaBancaria minhaConta = new ContaBancaria();`
 - `minhaConta.saldo = 50000;`
- O quê o java oferece para garantir a integridade dos atributos de uma classe?



Restrição de acesso

- Utilize a palavra reservada `private`;
- Com `private`, apenas métodos da própria classe podem acessar seus atributos:

```
private String numeroConta;  
private double saldo;  
private double limite;  
private Cliente titular;
```


Modificador de acesso - private

- **private** é um **modificador de acesso**
- Marcando um atributo como privado, fechamos o acesso ao mesmo de todas as outras classes, fazendo
- com que o seguinte código não compile:
- ```
class TestaAcessoDireto {
 ■ public static void main(String args[]) {
 ■ Conta minhaConta = new Conta();
 ■ //não compila! você não pode acessar o atributo privado de outra classe
 ■ minhaConta.saldo = 1000;
 ■ }
}
```



# Modificadores de acesso – private vs public

---

- O modificador public é utilizado para permitir que o componente fique acessível a qualquer classe;
- `private` e `public` podem ser utilizados para qualquer componente de uma classe: atributos ou métodos;



# Encapsulamento

---

- Começamos a ver, nos slides anteriores, a necessidade de encapsular o acesso ao atributos de uma classe;
- Com o encapsulamento de componente, facilitamos o processo de mudança de um sistema, uma vez que as regras de negócio serão alteradas apenas na classe;

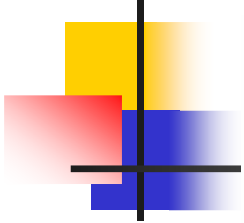


# Getters e Setters

---

- Para permitir o acesso aos atributos (já que eles são `private`) de uma maneira controlada, a prática mais comum é criar dois métodos, um que `retorna(GET)` o valor e outro que `muda(SET)` o valor.
- O padrão para esses métodos é de colocar a palavra `get` ou `set` antes do nome do atributo.
- Por exemplo, a nossa conta com saldo, limite e titular fica assim:

# Getters e Setters



```
public double getLimite() {
 return limite;
}

public void setLimite(double limite) {
 this.limite = limite;
}

public String getNumeroConta() {
 return numeroConta;
}

public void setNumeroConta(String numeroConta) {
 this.numeroConta = numeroConta;
}

public Cliente getTitular() {
 return titular;
}

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

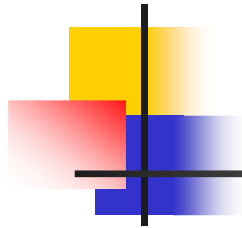
public void setSaldo(double saldo) {
 this.saldo = saldo;
}
}
```



# Construtores

---

- **Atualizar com mais teoria de construtores**
- Quando usamos a palavra chave **new**, estamos construindo um objeto. Sempre que **new** é chamado, executa o **construtor da classe**.
- O construtor da classe é um bloco declarado com o **mesmo nome** da classe:
- ```
public class ContaBancaria{  
    ■ Conta() {  
        ■ System.out.println("Construindo uma conta.");  
    }  
}
```



Tipos de construtor

- Toda classe java possui um construtor. Caso não exista, o java criará um construtor padrão;
- Em geral, os construtores são utilizados para a inicialização de atributos;
- Por isso, é possível criarmos construtores que recebem parâmetros do usuário;



Chamada de construtor

- Imagine que sempre precisamos informar o saldo inicial de uma nova conta:
- `ContaBancaria(double saldoInicial){`
 - `this.saldo = saldoInicial;`
 - `this.limite = 0; //inicia outros atributos`
- `}`
- Chamada: `ContaBancaria conta = new ContaBancaria(300.0);`



Atributos de classe

- Os atributos que foram apresentados, até aqui, pertencem às instâncias da classe. Ou seja, cada objeto altera seus valores;
- Como faríamos para controlar a quantidade de contas criadas, uma vez que cada objeto controla seus atributos?
- Seria interessante que a variável fosse única e compartilhada pelos objetos;

Declaração de atributos de classes



- Para isso, declaramos um atributo como static;
- `private static int totalDeContas;`
- Com isso, o atributo deixa de pertencer a cada o objeto e passa a ser atributo da classe ContaBancaria;
- Ainda que o atributo seja acessado por classes diferentes, o valor será o mesmo;



Atributos de classe

```
public class Conta {  
    private static int totalDeContas;  
    //declaracao do metodo construtor  
    Conta() {  
        Conta.totalDeContas++;  
    }  
    public int getTotalDeContas() {  
        return Conta.totalDeContas;  
    }  
    public void setTotalDeContas(int totalDeContas) {  
        Conta.totalDeContas = totalDeContas;  
    }  
    public static void main(String[] args) {  
        Conta a = new Conta();  
        System.out.println("Total de contas: "+a.getTotalDeContas());  
        Conta b = new Conta();  
        System.out.println("Total de contas: "+b.getTotalDeContas());  
    }  
}
```



Considerações:

- Em algumas empresas, a UML é amplamente utilizada. Às vezes, o programador recebe um diagrama UML pronto e só deve preencher a implementação. O que você acha dessa prática?
- Se uma classe só tem atributos e métodos estáticos, que conclusões podemos tirar? O que lhe parece um método estático?
- O padrão dos métodos get e set não vale para as variáveis de tipo boolean. Nesses casos, o **get** é substituído por **is**.
- Por exemplo, para verificar se um carro está ligado seriam criados os métodos **isLigado** e **setLigado**.



Exercícios de 01 a 04

- Crie a classe funcionário com: nome, matrícula e nome do departamento, utilizando `private`;
- Tente criar um objeto Funcionario no método `main()` e modificar ou ler um de seus atributos privados. O que acontece?
- Crie os getters e setters necessários à sua classe Funcionario;
- Faça com que sua classe Funcionario possa receber, opcionalmente, o nome do Funcionario durante a criação do objeto. Utilize `constructores` para obter esse resultado.



Exercícios 05 e 06

- Adicione um atributo à classe Funcionario de tipo int que se chama identificador. Esse identificador deve ter valor único para cada instância do tipo Funcionario. O primeiro Funcionario instanciado tem identificador 1, o segundo 2, e assim por diante. Você deve utilizar os recursos aprendidos aqui para resolver esse problema.
- Crie um getter para o identificador. Devemos ter um setter?



Bibliografia

- Java - Como programar, de Harvey M. Deitel
- Use a cabeça! - Java, de Bert Bates e Kathy Sierra
- (Avançado) Effective Java Programming Language Guide, de Josh Bloch



Referências WEB

- SUN: www.java.sun.com

Fóruns e listas:

- Javaranch: www.javaranch.com
- GUJ: www.guj.com.br

Apostilas:

- Argonavis: www.argonavis.com.br
- Caelum: www.caelum.com.br



Java Standard Edition (JSE)

Capítulo 05. Encapsulamento, Modificadores de acesso e atributos de classe



Esp. Márcio Palheta

MSN: marcio.palheta@hotmail.com