DIN Departamento de Informática

UEM Universidade Estadual de Maringá

Encapsulamento

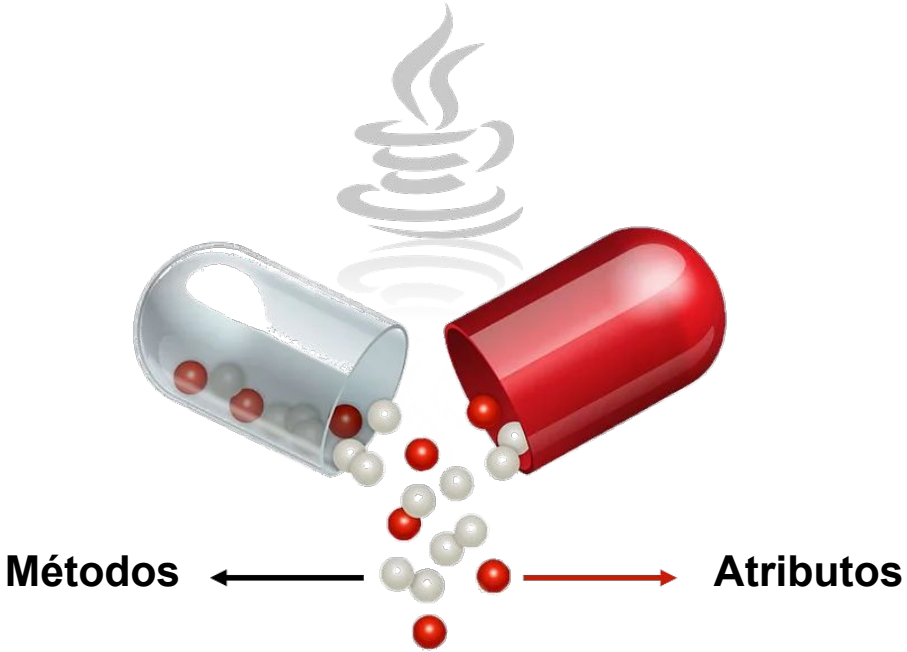
Lilian Passos Scatalon
lpscatalon2@uem.br

Encapsulamento

Envolve encapsular dados e comportamento em uma unidade de código

Os atributos e os métodos que trabalham sobre eles ficam na mesma classe

Encapsulamento



https://www.simplilearn.com/tutorials/java-tutorial/java-encapsulation

Encapsulamento

As variáveis de uma classe são acessadas pelos métodos da classe

Ficam ocultas a outras classes

Ocultamento de informação (*information hiding*)

Encapsulamento

As variáveis de uma classe são acessadas pelos métodos da classe

Ficam ocultas a outras classes

Ocultamento de informação (*information hiding*)

Modificadores de acesso

O modificador de acesso precede a declaração de um membro (atributo/método) da classe

```
public String mensagemErro;

private float saldo;
```

Modificadores de acesso

Public

Protected

Private

D*efault* (nenhuma palavra-chave é especificada)

Public

Membros públicos de uma classe podem ser acessados em qualquer parte do código (dentro e fora da classe)

```
public int dia, mes;
```

Public

```
public class Data {
    public int dia, mes;

        public void imprime(){
                System.out.print(dia + " de ");

                ...

        }
}

public class TesteData {
        public static void main(String[] args) {
                Data hoje = new Data();

                hoje.dia = 23;
                hoje.mes = 1;

                hoje.imprime();
        }
}
```

dia e mes são atributos públicos da classe Data

Podem ser acessados dentro da classe Data…

… e fora da classe Data
Aqui o acesso é feito a partir da classe TesteData

Public

```
public class Data {
    public int dia, mes;

        public void imprime(){
                System.out.print(dia + " de ");

                ...

        }
}

public class TesteData {
        public static void main(String[] args) {
                Data hoje = new Data();

                hoje.dia = 23;
                hoje.mes = 1;

                hoje.imprime();
        }
}
```

imprime() é método público da classe Data

Pode ser acessado (chamado) fora da classe Data, em TesteData

Private

Membros privados apenas podem ser acessados por métodos da mesma classe

O acesso é negado para código que esteja fora da classe (métodos de outras classes)

```
private int dia, mes;
```

Private

```
public class Data {
    private int dia, mes;

        public void imprime(){
                System.out.print(dia + " de ");

                ...

        }
}

public class TesteData {
        public static void main(String[] args) {
                Data hoje = new Data();

                hoje.dia = 23;  // erro!
                hoje.mes = 1;   // erro!

                hoje.imprime();
        }
}
```

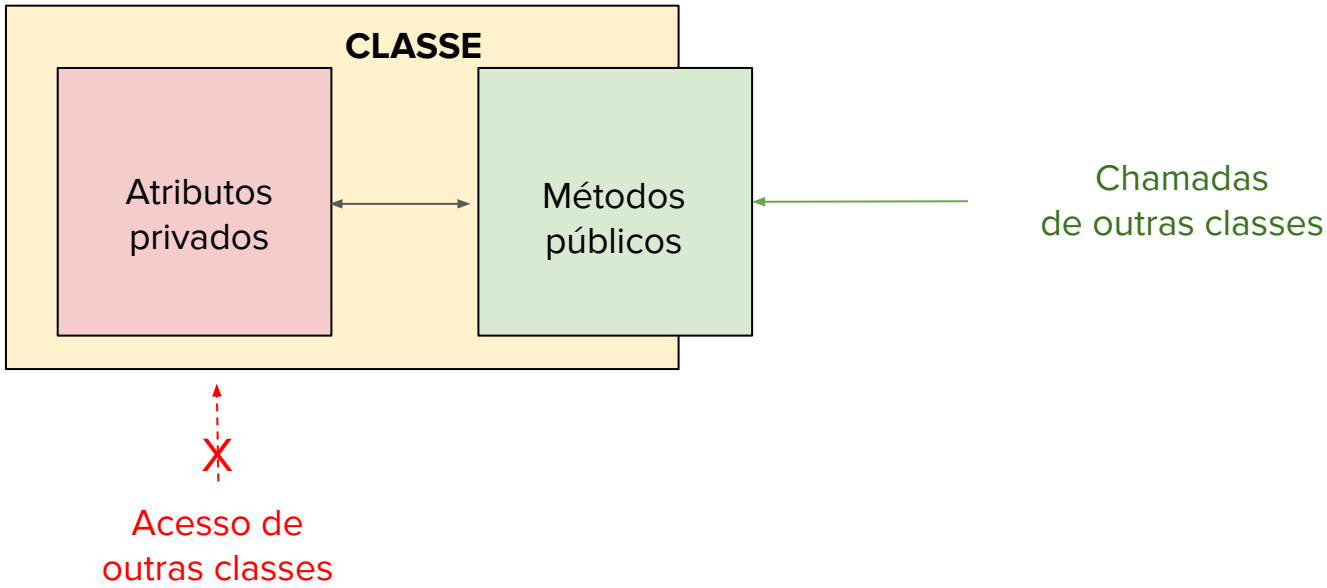
dia e mes são atributos **privados** da classe Data

Podem ser acessados dentro da classe Data

Mas não podem ser acessados em TesteData, pois está fora da classe Data

Getters e setters

Getters e setters são métodos públicos auxiliares para manipular atributos privados



Getter

Método público que permite consultar/obter um atributo privado

Nome: prefixo get + nome do atributo

Consulta o valor de um atributo

```
public int getDia() {
    return dia;
}
```

Setter

Método público que permite alterar o valor de um atributo privado

Nome: prefixo set + nome do atributo

Permite a verificação de entradas inválidas

```
public void setDia(int dia) {
    if(dia>=1 || dia<=31)
        this.dia = dia;
}
```

Encapsulamento

Raciocínio para determinar os modificadores de acesso:

- atributos privados, a menos que haja motivo para permitir o acesso por outras classes
- métodos públicos, a menos que haja motivo para limitar a chamada do método por outras classes

Modificadores de acesso

	Dentro da mesma classe	Em uma classe derivada	Fora da classe
Private	Sim	Não	Não
Protected	Sim	Sim	Não
Public	Sim	Sim	Sim

Modificadores de acesso

	Dentro da mesma classe	Fora da classe
Private	Sim	Não
Public	Sim	Sim

Interface e Implementação

Pelo princípio do encapsulamento, um programador que usa uma classe não deve se preocupar com detalhes de como a classe é implementada

É preciso saber apenas “as regras” de como usar a classe

Essas “regras” são conhecidas como interface ou API (*Application Programming Interface*)

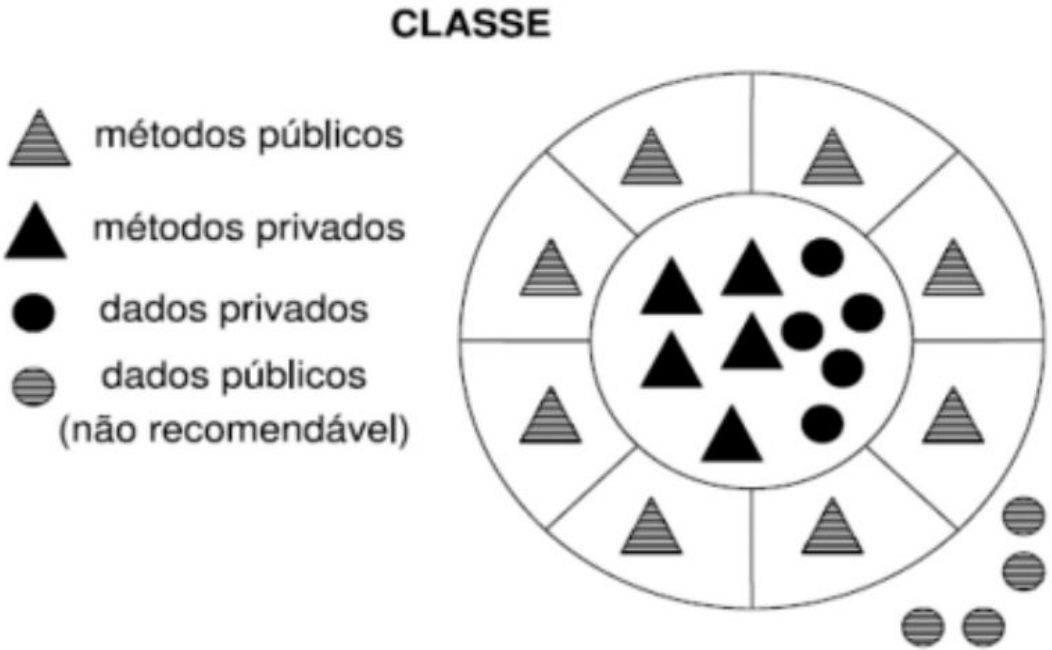
Interface e Implementação

Interface: atributos e métodos públicos da classe

Implementação: atributos privados e a definição dos métodos públicos e privados

Em uma classe bem projetada, é suficiente saber apenas a sua interface para conseguir usá-la em outros programas.

Encapsulamento



Interface e Implementação - exemplo

Classe Agenda

Interface:

- public Agenda(int tamanhoMaximo)
- public void adicionaContato(Pessoa p)
- public void removeContato(String nome)
- public void imprimeAgenda()

métodos públicos
(a classe Agenda não tem atributos públicos)

Interface e Implementação - exemplo

Classe Agenda

Implementação:

- private int tamanhoMaximo;
- private int posicaoAtual;
- private Pessoa[] contatos;
- private int buscaContato(String nome)

atributos e métodos privados

Interface e Implementação - exemplo

Classe Agenda

Implementação:

definição dos métodos

```
public class Agenda {
        private int tamanhoMaximo;
        private int posicaoAtual;
        private Pessoa[] contatos;

        public Agenda(int tamanhoMaximo) {
                this.tamanhoMaximo = tamanhoMaximo;
                this.posicaoAtual = 0;
                contatos = new Pessoa[tamanhoMaximo];
        }

        public void adicionaContato(Pessoa p) {
                if(posicaoAtual<tamanhoMaximo) {
                        contatos[posicaoAtual] = p;
                        posicaoAtual++;
                }
                else {
                        System.out.println("Capacidade máxima atingida." +
                                        "Não foi possível adicionar o contato.");
                }
        }
```

Interface e Implementação - exemplo

Classe Agenda

Implementação:

definição dos métodos

```
public void removeContato(String nome) {
        int posicao = buscaContato(nome);

        if(posicao == -1)
                return;

        // fazer os deslocamentos para não deixar "buracos" no vetor
        // "shift"
        for(int i=posicao; i<posicaoAtual; i++) {
                contatos[i] = contatos[i+1];
        }

        posicaoAtual--;

}

private int buscaContato(String nome) {

        if(posicaoAtual == 0) {
                System.out.println("Agenda vazia.");
                return -1;
        }

        for(int i=0; i<posicaoAtual; i++) {
                Pessoa p = contatos[i];
                if(p.getNome().equals(nome))
                        return i;
        }

        System.out.println("Não existe contato com o nome informado.");
        return -1;
}
```

Interface e Implementação - exemplo

Classe Agenda

Implementação:

definição dos métodos

```
public void imprimeAgenda() {
        for(int i=0; i<posicaoAtual; i++) {
                Pessoa p = contatos[i];
                System.out.println(p);
        }
}
```

Conceitos de POO

1. Encapsulamento
2. Abstração
3. Herança
4. Polimorfismo

Bibliografia

Deitel, P.J. e Deitel, H.M. **Java How to Program: Late Objects**. 11th edition. Pearson, 2020.

C. S. Horstmann. **Core Java SE 9 for the Impatient**. Addison-Wesley, 2nd Edition, 2018.

Barnes, D. e Kolling, M. **Objects First with Java: A Practical Introduction Using BlueJ**. Pearson, 2016.

Savitch, W. **Absolute C++.** Addison Wesley, 2001.