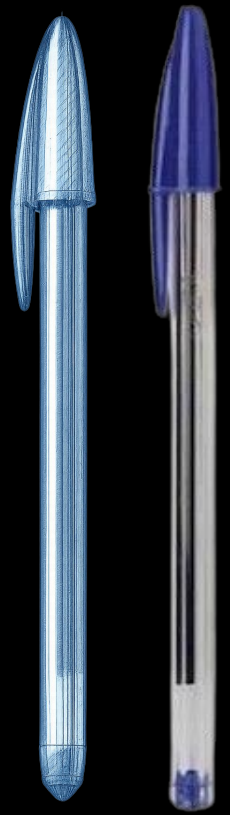
Visibilidade e Encapsulamento

Matheus Barbosa
matheus.barbosa@dcx.ufpb.br

UML - Diagrama de classe

Caneta
modelo: String cor: String ponta: float carga: int tampada: boolean
rabiscar() escrever() tampar() destampar()

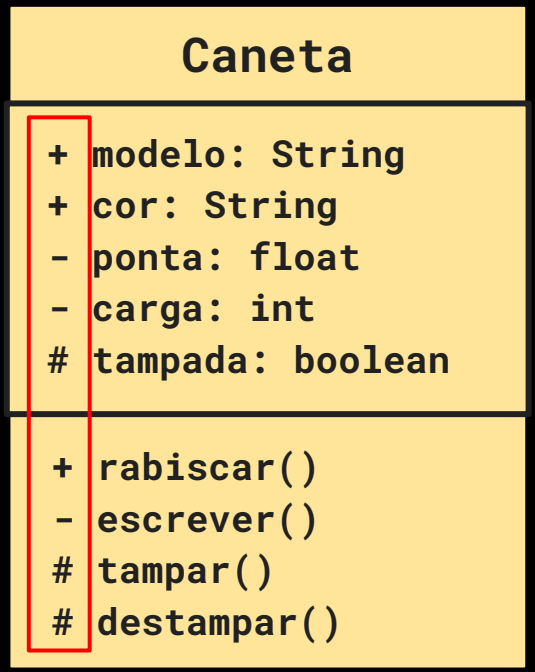
```
Classe Caneta
   modelo: texto
   cor: texto
   ponta: decimal
   carga: inteiro
   tampada: logico

   rabiscar()
   escrever()
   tampar()
   destampar()

FimClasse
```



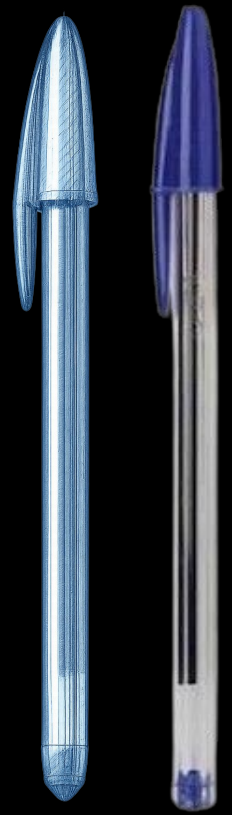
Visibilidade Indica o nível de acesso aos componentes internos de uma classe

Símbolo	Modificador	Descrição
+	Público	A classe atual e todas as outras classes
-	Privado	Somente a classe atual
#	Protegido	Classes no mesmo pacote e subclasses
	(Sem modificador - "default" ou "package-private")	Apenas classes no mesmo pacote

UML - Sinais de visibilidade




```
Classe Caneta
   publico modelo: texto
   publico cor: texto
   privado ponta: decimal
   privado carga: inteiro
   protegido tampada: logico

   publico rabiscar()
   privado escrever()
   protegido tampar()
   protegido destampar()
FimClasse
```



Métodos inicializadores

```
class Caneta {
  public String modelo;
  public String cor;
  …
  Ceneta(String modelo, String cor){
    modelo = modelo;
    cor = cor;
  }
  …
}
```

Além de métodos e atributos, a definição de uma classe pode incluir também a definição de inicializadores (também chamados construtores, apesar de não construírem nada!) que são operações que podem ser utilizadas para inicializar os atributos dos objetos

Métodos inicializadores

```
class Caneta {
  public String modelo;
  public String cor;
  …
  Ceneta(String modelo, String cor){
    modelo = modelo;
    cor = cor;
  }

  Ceneta(String modelo){
    modelo = modelo;
  }
  …
}
```

Inicializadores têm o mesmo nome da classe, podendo haver mais de um desde que com número e/ou tipos de argumentos diferentes

```
class Main {
 public main(){
   Caneta c1 = new Caneta("M1", "Azul");
   Caneta c2 = new Caneta("M2");
  }
}
```

Getters e Setters

São métodos que permitem o acesso indireto a atributos privados de uma classe.

Em um contexto ideal, todos os atributos de uma classe são mantidos privados para protegê-los de acessos e modificações não autorizadas. Para acessar ou modificar esses atributos, são utilizados métodos públicos conhecidos como *getters* e *setters*.

Getters

```
class Caneta {
  private String modelo;
  private String cor;
  …
  public getModelo(){
    return modelo;
  }

  public getCor(){
    return cor;
  }
  …
}
```

Os *getters* são métodos públicos que permitem a leitura dos valores dos atributos. Eles não modificam os dados, apenas retornam seus valores.

A convenção em Java é nomear esses métodos com o prefixo "*get*" seguido pelo nome do atributo com a primeira letra em maiúscula.

Setters

```
class Caneta {
  private String modelo;
  private String cor;
  …
  public setModelo(String modelo){
    modelo = modelo;
  }

  public setCor(String cor){
    cor = cor;
  }
  …
}
```

Os *setters*, são usados para modificar os valores dos atributos. Eles são definidos com o prefixo "*set*", seguido pelo nome do atributo com a primeira letra maiúscula. Esses métodos geralmente aceitam um parâmetro que é usado para atualizar o valor do atributo.

Melhores Práticas

- **Validação de Dados:** No *setter*, sempre valide os dados antes de atribuí-los aos atributos. Isso ajuda a manter a integridade dos dados da classe.

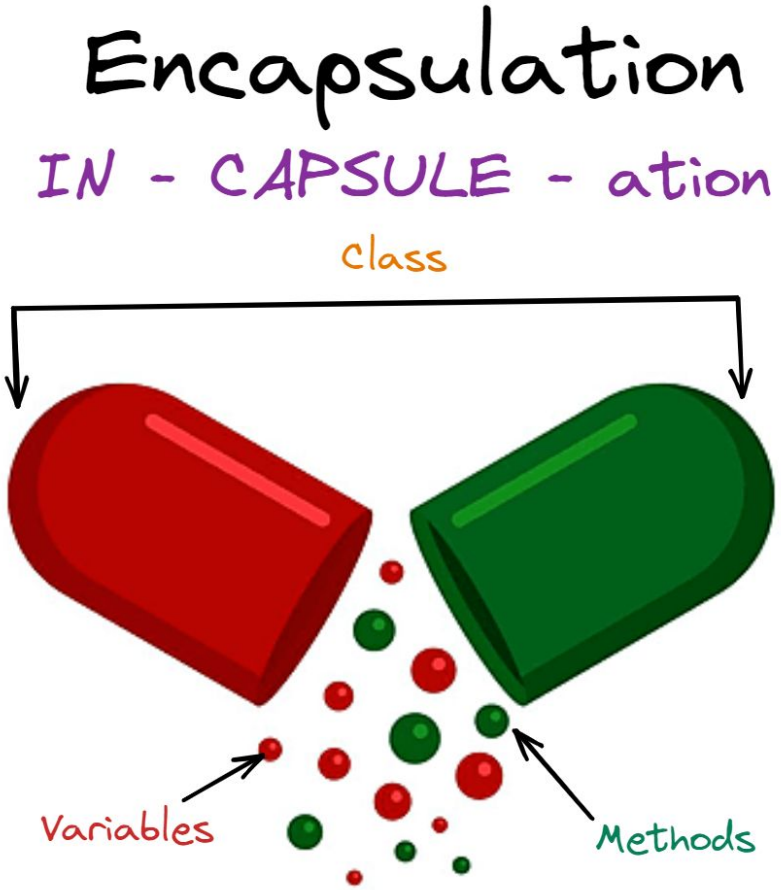
- **Nominação Consistente:** Utilize convenções de nomenclatura consistentes para que os *getters* e *setters* sejam facilmente reconhecíveis e utilizáveis por outros desenvolvedores.

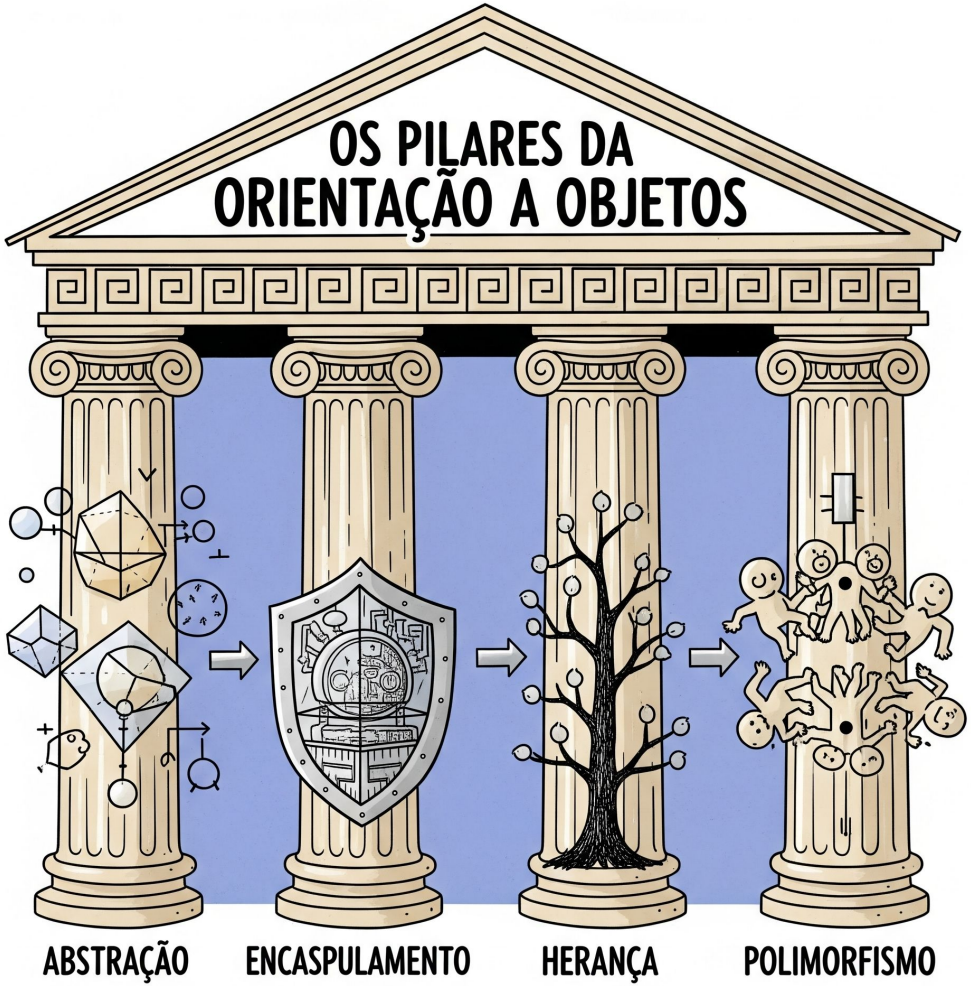
- **Minimizar o Uso de *Setters* para Atributos Imutáveis:** Para atributos que não devem ser alterados após a criação do objeto, é recomendável não implementar *setters*. Em vez disso, os valores podem ser definidos uma única vez através do construtor.

Getters e *setters* são mais do que simples métodos em Java; eles são uma manifestação prática do princípio de encapsulamento. Eles não só oferecem uma maneira segura e controlada de acessar os dados de um objeto, mas também promovem uma arquitetura de software robusta, flexível e fácil de manter.

Encapsulamento

Consiste em ocultar os detalhes internos de uma classe, expondo para os demais objetos somente os dados e métodos necessários por meio de interfaces públicas.




OS PILARES DA
ORIENTAÇÃO A OBJETOS
ABSTRAÇÃO
ENCASPULAMENTO
HERANÇA
POLIMORFISMO

Interfaces públicas.

Essas interfaces públicas seriam dados ou métodos que podem ser acessados/chamados por outros objetos.

Os métodos públicos são as ações que um objeto expõe para o mundo externo, definindo o que ele é capaz de fazer sem revelar como faz.

Benefícios do Encapsulamento

1. **Controle de Acesso:** Permite que a classe controle como as variáveis de instância são acessadas ou modificadas, garantindo, por exemplo, que valores inválidos não sejam atribuídos.
2. **Flexibilidade e Manutenção:** A implementação da classe pode ser alterada sem impactar as classes que a utilizam, desde que os métodos *getters* e *setters* sejam mantidos.
3. **Segurança de Dados:** Protege a integridade dos dados, impedindo que sejam expostos ou alterados de forma inadequada.