O que são Getters e Setters?

Getters e **setters** são métodos usados para acessar e modificar os atributos de uma classe. Eles são parte do princípio do **encapsulamento**, que é uma das características fundamentais da POO.

- **Getter**: É um método que permite acessar o valor de um atributo privado de uma classe. Ele geralmente segue a convenção de nomeação `getNomeDoAtributo()`. Por exemplo, se você tiver um atributo `idade`, o getter seria `getIdade()`.
- **Setter**: É um método que permite modificar o valor de um atributo privado. O nome do setter geralmente segue a convenção `setNomeDoAtributo()`. Por exemplo, para o atributo `idade`, o setter seria `setIdade(int idade)`.

Importância do Encapsulamento

O uso de getters e setters traz várias vantagens:

1. **Controle de Acesso**: Eles permitem controlar como os atributos de uma classe são acessados e modificados. Você pode, por exemplo, adicionar validações nos setters para garantir que os dados sejam válidos antes de serem atribuídos.
2. **Facilidade de Manutenção**: Se a implementação interna da classe mudar, você pode ajustar apenas os métodos getters e setters sem afetar o código que utiliza a classe.
3. **Leitura e Escrita Simples**: Eles tornam o código mais legível, pois ao usar `pessoa.getNome()`, fica claro que você está acessando um nome e não apenas um atributo.

Exemplo Prático

Aqui está um exemplo de uma classe `Pessoa` com getters e setters:

```
public class Pessoa {
    private String nome;
    private int idade;

    // Getter para nome
    public String getNome() {
        return nome;
    }
```

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

    // Getter para idade
    public int getIdade() {
        return idade;
    }

    // Setter para idade com validação
    public void setIdade(int idade) {
        if (idade >= 0) { // Validação simples
            this.idade = idade;
        } else {
            throw new IllegalArgumentException("Idade não pode ser
negativa.");
        }
    }
}
```

O que são Construtores?

Um **construtor** é um método especial em uma classe que é chamado automaticamente quando um objeto daquela classe é criado. O principal objetivo do construtor é inicializar os atributos do objeto. Os construtores têm o mesmo nome da classe e não têm tipo de retorno (não retornam void ou qualquer outro tipo).

Tipos de Construtores

1. **Construtor Padrão**: É um construtor que não aceita parâmetros. Se você não definir nenhum construtor em sua classe, o compilador cria automaticamente um construtor padrão que inicializa os atributos com valores padrão (como null para objetos, 0 para números, etc.).

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

```
    // Construtor padrão
    public Carro() {
        this.marca = "Desconhecida";
        this.modelo = "Desconhecido";
        this.ano = 0;
    }
}
```

2. **Construtor Parametrizado**: Este construtor aceita um ou mais parâmetros, permitindo que o usuário forneça valores para os atributos do objeto no momento da criação.

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

    // Construtor parametrizado
    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }
}
```

Sobrecarregando Construtores

Os construtores podem ser **sobrecarregados**, o que significa que você pode ter mais de um construtor na mesma classe, desde que eles tenham listas de parâmetros diferentes. Isso oferece flexibilidade na criação de objetos.

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

    // Construtor padrão
    public Carro() {
        this("Desconhecida", "Desconhecido", 0);
    }
```

```
    // Construtor parametrizado
    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }

    // Outro construtor com menos parâmetros
    public Carro(String marca, String modelo) {
        this(marca, modelo, 2022); // Chama o construtor com ano
padrão
    }
}
```

Importância dos Construtores

- **Inicialização de Atributos**: Os construtores garantem que os objetos sejam criados em um estado válido, com seus atributos inicializados corretamente.
- **Flexibilidade**: Com construtores parametrizados e sobrecarga, você pode criar objetos de diferentes maneiras, dependendo das necessidades do seu programa.
- **Facilidade de Uso**: Eles tornam a criação de objetos mais intuitiva, pois você pode passar valores diretamente no momento da instância.

Questões sobre Getters e Setters

1. **O que são getters e setters em POO?**
    a. Getters e setters são métodos que permitem acessar e modificar os atributos privados de uma classe. Getters retornam o valor de um atributo, enquanto setters definem ou alteram esse valor. Eles são importantes para manter o encapsulamento, permitindo que o controle sobre a manipulação dos dados seja feito.
2. **Por que é recomendado usar getters e setters em vez de acessar diretamente os atributos de uma classe?**

a. O uso de getters e setters promove o encapsulamento, permitindo que a implementação interna da classe seja alterada sem afetar o código que a utiliza. Isso facilita a manutenção, testes e o controle de acesso a dados, permitindo também validações ao modificar atributos.

3. **Como você implementaria um getter e um setter em uma classe `Pessoa` que possui um atributo `nome`?**

```
public class Pessoa {
    private String nome;

    public String getNome() {
        return nome;
    }

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

4. **Qual é a diferença entre um getter e um setter em termos de retorno e parâmetros?**
    a. Um getter não possui parâmetros e retorna o valor de um atributo, enquanto um setter recebe um parâmetro (o novo valor) e não retorna nada. O getter é usado para obter o estado do objeto, e o setter para modificá-lo.
5. **Quais são algumas práticas recomendadas ao usar setters?**
    a. Valide o valor antes de atribuí-lo ao atributo (por exemplo, checar se um número está dentro de um intervalo). Além disso, é bom manter o setter simples e claro, e evitar realizar operações complexas dentro dele.

Questões sobre Construtores

6. **O que é um construtor em POO e qual é sua principal finalidade?**
    a. Um construtor é um método especial chamado quando um objeto é instanciado. Sua principal finalidade é inicializar os atributos do objeto com valores fornecidos ou padrão, garantindo que o objeto comece em um estado válido.
7. **Como você definiria um construtor em uma classe chamada `Carro` que recebe `marca`, `modelo` e ano como parâmetros?**

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

    public Carro(String marca, String modelo, int ano) {
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }
}
```

8. **Qual é a diferença entre um construtor padrão e um construtor parametrizado?**
    a. Um construtor padrão não aceita parâmetros e é usado para inicializar atributos com valores padrão. Um construtor parametrizado aceita um ou mais parâmetros, permitindo a inicialização dos atributos com valores específicos. Exemplo de construtor padrão:

```
public Carro() {
    this.marca = "Desconhecida";
    this.modelo = "Desconhecido";
    this.ano = 0;
}
```

9. **Como os construtores podem ser sobrecarregados? Dê um exemplo.**
    a. Construtores podem ser sobrecarregados ao ter o mesmo nome, mas com diferentes listas de parâmetros. Exemplo:

```
public Carro(String marca, String modelo, int ano) {
    this.marca = marca;
    this.modelo = modelo;
    this.ano = ano;
}

public Carro(String marca, String modelo) {
    this(marca, modelo, 2022); // chama o construtor acima com um
ano padrão
}
```

10. O que acontece se você não definir um construtor em uma classe?

a. Se nenhum construtor for definido, o compilador fornece um construtor padrão que não aceita parâmetros. Esse construtor inicializa os atributos com valores padrão (null para objetos, 0 para inteiros, etc.). No entanto, se você definir um construtor personalizado, o construtor padrão não será gerado automaticamente.