**POO - Objetivo:** Melhorar a estrutura do código, organizar, reutilizar e facilitar a manutenção do sistema. **Modularização:** Quebrar o código em partes menores e direcioná-las para o sistema (abstração). **Abstração:** Representar o mundo real no sistema. **Atributos:** Representam características. **Métodos:** Representam ações.

**Classes - Definição:** Representação de um objeto dentro do código, com regras dentro da classe. **Nome:** Sempre começa com letra maiúscula e no singular (ex: Carro). **Instanciação:** Criar um objeto novo usando new NomeDaClasse(), alocando espaço na memória. **Variável reference:** Variável auxiliar no código, indicada como m\_variavel.

**Construtor  
Definição:** Obriga a inicializar um objeto com argumentos específicos. **Construtor Implícito:** Criado automaticamente pelo Java com atributos padrões. **Construtor Explícito:** Criado pelo programador. **Regras: 1)** Não possuem void ou retorno. **2)** Não pode haver construtores com a mesma assinatura (causa erro). **Referência:** Usar this para referenciar a própria classe.

public Pessoa() {

this("Usuario");

}

**Encapsulamento - Objetivo:** Proteger os atributos da classe, as coisas definidas passam ser exclusivas da classe, e não mais de objetos externos a ela.  
**Modificadores de Acesso:** **public:** Acesso por qualquer classe. - **private:** Acesso apenas dentro da classe. - **protected:** Acesso dentro da classe e suas subclasses.  
Métodos privados encapsulam funcionalidades que são utilizadas apenas dentro da classe, mas não podem ser acessados por outras classes ou partes do código. Esses métodos são conhecidos como métodos auxiliares.

**Getter:** Método para leitura dos atributos, sempre retorna o tipo do dado. | **Setter:** Método para alterar os atributos, sem retorno (void), usando um parâmetro com o valor a ser modificado. Para atributos booleanos, use isVariavel em vez de getVariavel.

**Evitar Acesso Direto:** É uma má prática acessar diretamente os atributos com p.nome.  
**Verificações Internas:** Use getters e setters para validar alterações dentro dos métodos.

**Associação:** Vínculo entre objetos, onde um objeto faz referência a outro.  
**Exemplo:** Uma disciplina se relaciona com um professor.  
Uma associação é uma relação de posse, um objeto possui um objeto, ou seja, uma disciplina tem sempre um único professor.  
O objeto Professor (ou o objeto que foi criado dentro de outro) como a Disciplina, pode ter os atributos e métodos públicos acessados dentro da Classe no qual o objeto foi criado.  
resumo: quando vc cria um objeto dentro do outro, esse objeto criado vira um “atributo”

|  |  |  |
| --- | --- | --- |
|  | // Acessando o tamanho do array int tamanho = valores.length; // 5  // Usando um loop for tradicional for (int i = 0; i < valores.length; i++) {  System.out.println("Elemento na posição " + i + ": " + valores[i]); }  Usando um loop for-each for (int valor : valores) {  System.out.println("Valor: " + valor); } | // Declaração de um array de inteiros com 5 posições int[] numeros = new int[5];  // Inicializando um array com valores int[] valores = {1, 2, 3, 4, 5}; // Acessando elementos do array int primeiroValor = valores[0]; // 1 int segundoValor = valores[1]; // 2 // Modificando um elemento do array valores[2] = 10; // O terceiro elemento agora é 10 |

**Coleção:** Estrutura que armazena múltiplos objetos. Exemplo: ArrayList, LinkedList.

|  |  |  |
| --- | --- | --- |
| **Apesar de um array ser capaz de armazenar vários objetos, ele não é considerado um tipo de coleção.**  **Coleções possuem funcionalidades que podem ser** | **utilizadas para simplificar alguns processos que são mais trabalhosos quando utilizamos arrays.** |  |

**Array:** Estrutura estática, com tamanho fixo. Não é uma coleção, mas pode armazenar objetos.  
**Estático:** O tamanho é definido na criação e não pode ser alterado.  
**Manipulação:** Copiar dados para um novo array maior é a única forma de expandir.

Texto

Descrição gerada automaticamente