

The background features several decorative elements: a large white circle in the top-left corner, a solid purple circle in the top-left area, a purple horizontal bar in the top-right corner, a small purple circle in the bottom-right area, a large white circle in the bottom-right corner, and a purple horizontal bar in the bottom-left corner.

Programação II: Orientação a Objetos

Prof^a. Tainá Isabela

Conceitos Básicos

Na visão da orientação a objetos, o mundo é composto por diversos objetos que possuem um conjunto de características e um comportamento bem definido.

- Estado: conjunto de propriedades de um objeto (valores dos atributos).
- Comportamento: conjunto de ações possíveis sobre o objeto (métodos da classe).
- Unicidade: todo objeto é único (possui um endereço de memória).

Classe

Podemos entender uma classe como um modelo ou como uma especificação para um **conjunto de objetos**, ou seja, a descrição genérica dos objetos individuais pertencentes a um dado conjunto. A partir de uma classe é possível criar quantos objetos forem desejados.

Uma **classe** define as características e o comportamento de um conjunto de objetos. Assim, a criação de uma classe implica definir um tipo de objeto em termos de seus atributos (variáveis que conterão os dados) e seus métodos (funções que manipulam tais dados).

Classe Pessoa

id
nome
idade
cidade
gênero

cadastrar()
getNome()
setNome()
getSexo()
setSexo()
...
...

Objeto 1



id = 1
nome = "Marcelo"
cidade = Cuiabá
gênero = M

Objeto 2



id = 2
nome = "Maria"
cidade = Goiânia
gênero = F

Objeto

É a criação de uma **instância da classe**. Quando instanciamos um objeto, criamos fisicamente uma representação concreta de sua classe.

“Ser Humano” é uma classe, um molde, já “Roberto” é uma instância de “Ser Humano”. Apesar de carregar todas as características do molde “Ser Humano”, ele é completamente diferente (independente) das outras instâncias de “Ser Humano”.

Pacotes

Em Java são uma maneira de **agrupar classes** e interfaces relacionadas.

Pacotes permitem que grupos de classes estejam disponíveis apenas se forem necessários e eliminam possíveis conflitos entre nomes de classes em diferentes grupos de classes.

Quando desenvolvemos um **sistema orientado a objetos**, definimos um conjunto de classes. Para organizar essas classes surge o conceito de **pacote**. Pacote é um envoltório de classes, ou seja, guarda classes e outros pacotes logicamente semelhantes ao pacote que os contém.

Classes em java

Sintaxe:

```
<qualificador> class <nome_da_classe>{  
  
    <declaração dos atributos da classe>  
  
    <declaração dos métodos da classe>  
  
}
```

Como funciona?

<qualificador>: O qualificador de acesso determinará a visibilidade da classe. Pode ser *public* (classe pública) ou *private* (classe privada). Classes privadas só poderão ser visualizadas dentro de seu próprio pacote enquanto as públicas serão acessíveis por qualquer classe de qualquer pacote. Se o qualificador for omitido, a classe será privada por padrão.

<nome>: nome que identifica a classe. Há um padrão entre os programadores de sempre iniciarem os nomes de classes com letras maiúsculas. Mas, apesar de ser uma boa prática, seguir esse padrão não é uma obrigação.

```
9 import pacote1.ClassePublica;
10 import pacote1.ClassePrivada;
11
12
13 public class ClasseOutroPacote
14 {
15
16
17     ClassePublica obj1;
18     ClassePrivada obj2;
```


Declaração de atributos e métodos

Sintaxe:

```
<qualificador> <tipo_do_atributo> <nome_do_atributo>;
```

Como funciona?

<qualificador>: O qualificador de acesso determinará a visibilidade do atributo. É opcional e, se não for informado, por padrão o atributo será protegido (*protected*). Não se preocupe com isso agora. Voltaremos a falar sobre os qualificadores quando estudarmos encapsulamento.

<tipo_do_atributo>: é um tipo primitivo ou classe que define o atributo.
<nome_do_atributo>: nome que identifica o atributo. Há um padrão entre os programadores de sempre iniciarem os nomes de atributos com letras minúsculas. Mas, apesar de ser uma boa prática, seguir esse padrão não é uma obrigação. Caso queiramos definir vários atributos de mesmo tipo podemos colocar os vários nomes separados por vírgula.

```
public class Conta
{
    int numero;
    String nome_titular;
    double saldo;

    void depositar(double valor) {
        this.saldo = this.saldo + valor;
    }
}
```

Sentindo falta de algo?

Utilização de Objetos

- **Declarar uma variável que referenciará o objeto:** assim como fazemos com tipos primitivos, é necessário declarar o objeto.
- **Instanciar o objeto:** alocar o objeto em memória. Para isso utilizamos o comando new e um construtor.

```
1 package exemplos;
2 public class Programa
3 {
4     public static void main(String[] args){
5
6         Conta c;
7         c = new Conta();
8         c.nome_titular="Jao";
9         System.out.println("Titular: "+c.nome_titular);
10        System.out.println("Saldo Atual: "+c.saldo);
11    }
12 }
```

```
1 package exemplos;
2 public class Programa
3 {
4     public static void main(String[] args){
5
6         Conta c = new Conta();
7         c.depositar(200);
8         boolean saque_efetuado = c.sacar(250);
9         if (saque_efetuado)
10             System.out.println("Saque Efetuado com Sucesso!");
11         else
12             System.out.println("Saque nao efetuado! Saldo insuficiente!");
13     }
14 }
```

Atributos e métodos estáticos

Atributos estáticos são atributos que contêm informações **inerentes** a uma classe e não a um objeto em específico. Por isso são também conhecidos como atributos ou variáveis de classe.


O mesmo conceito é válido para métodos. Métodos estáticos são inerentes à classe e, por isso, não nos obrigam a instanciar um objeto para que possamos utilizá-los. Para definir um método como estático, basta utilizar a palavra **static**, a exemplo do que acontece com atributos.



Classe String

Para criar uma instância de **String**, não precisamos utilizar o operador `new`, como acontece com as outras classes. Para instanciar um objeto do tipo `String`, basta declarar uma variável desse tipo e iniciá-la com um valor. É importante saber também que objetos da classe `String` podem ser concatenados utilizando o operador `+`.

Para comparar se os valores de duas `Strings` são iguais, utilizamos o método `"equals"` e não o operador `"=="` que é utilizado para tipos primitivos.



length: retorna o tamanho (tipo *int*) de uma *String*.

charAt: retorna o *character* (*char*) da *String* que se localiza no índice passado como parâmetro. Vale ressaltar que o primeiro índice de uma *String* é o índice zero.

toUpperCase: retorna uma *String* com todas as letras maiúsculas a partir da *String* que chamou o método.

toLowerCase: retorna uma *String* com todas as letras minúsculas a partir da *String* que chamou o método.

trim: retorna uma *String* sem espaços em branco no início e no final dela, a partir da *String* que chamou o método.

replace: Retorna uma *String* com *substrings* trocadas, a partir da *String* que chamou o método. As trocas são feitas de acordo com os parâmetros do método: em que aparecer a *substring1* será substituída pela *substring 2*.

valueOf: retorna uma *String* a partir de um valor de outro tipo, como um número por exemplo.


```
1 package exemplos;
2
3
4 public class ExemploString
5 {
6     public static void main(String[] args){
7         String frase1= " Muito a aprender ";
8         String frase2= " Voce ainda tem ";
9         String completa= frase1 + frase2;
10        System.out.println(completa + " !");
11        System.out.println("O caracter da posição 2 da frase 1 eh: "+frase1.charAt(2));
12        System.out.println("Frase completa toda em maiuscula: "+frase1.toUpperCase());
13        System.out.println("Substring de 2 a 8: " +completa.subSequence(2,8));
14        System.out.println("Tirando os espaços antes e depois da frase completa: "+completa.trim());
15        System.out.println("Substituindo aprender por praticar na frase completa: "+completa.replace("aprender", "praticar"));
16    }
17 }
18
19 }
```

Muito a aprender voce ainda tem !

O caracter da posicao 2 da frase 1 eh: i

Frase completa toda em maiuscula: MUITO A APRENDER

Substring de 9 a 15: prende

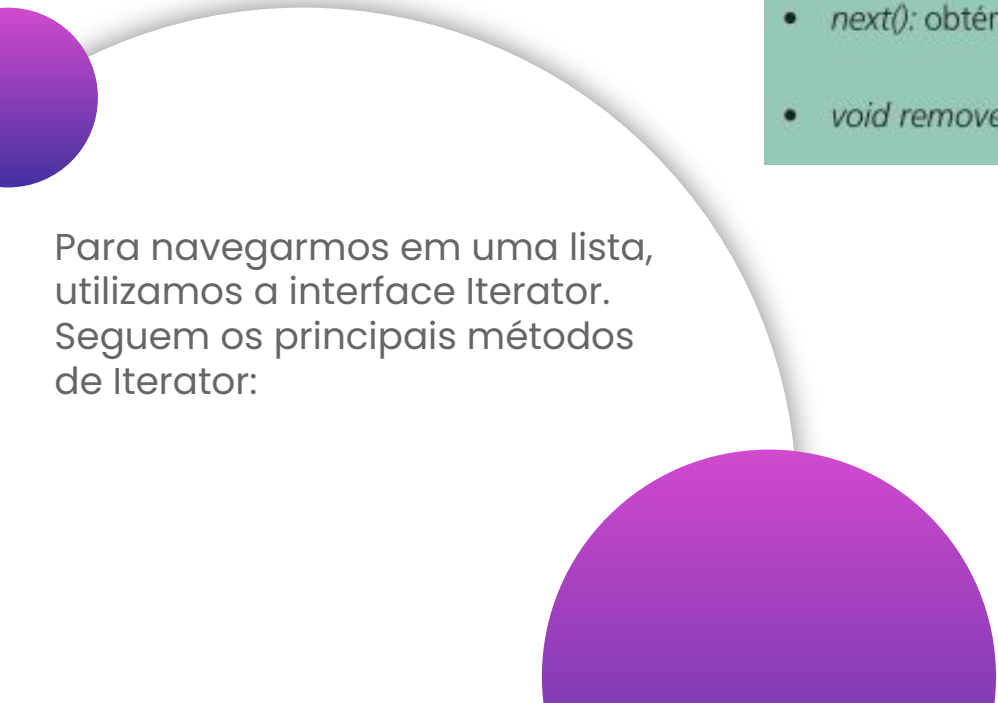
Tirando os espacos antes e depois da frase completa: Muito a aprender voce ainda tem

Substituindo aprender por praticar na frase completa: Muito a praticar voce ainda tem

Listas

A estrutura de dados lista é utilizada para armazenar conjuntos de elementos. A vantagem em utilizar listas em lugar de vetores é o fato de as listas serem alocadas dinamicamente de forma que não precisamos prever seu tamanho máximo. Java fornece classes que implementam o conceito de lista como o **ArrayList**.

- `public ArrayList()`: cria um `ArrayList` vazio.
- `public boolean add(<elemento>)`: adiciona um elemento no final da lista.
- `public void add(index, <elemento>)`: adiciona um elemento na posição `index`.
- `public <elemento> get(int index)`: obtém o elemento de índice `index`.
- `public <elemento> remove(int index)`: retorna o elemento de índice `index` e o elimina da lista.
- `public boolean isEmpty()`: verifica se a lista está vazia.



Para navegarmos em uma lista, utilizamos a interface `Iterator`. Seguem os principais métodos de `Iterator`:

- `boolean hasNext()`: verifica se existe próximo elemento na lista;
- `next()`: obtém o elemento sob o *Iterator* e avança para o próximo elemento;
- `void remove()`: remove o elemento sob o *Iterator*.

```
1 package exemplos;
2
3 import java.util.ArrayList;
4 import java.util.Iterator;
5
6
7 public class ExemploLista
8 {
9     public static void main(String[] args){
10         ArrayList lista = new ArrayList ();
11         Conta c = new Conta ();
12         c.numero = 1;
13         lista.add(c);
14         c = new Conta();
15         c.numero = 2;
16         lista.add(0,c);
17         Iterator i = lista.iterator();
18         while (i.hasNext()){
19             c=(Conta)i.next();
20             System.out.println("Conta Numero: "+c.numero);
21         }
22     }
23 }
```

Momento Questionário



Atividades

1. Crie uma classe chamada Carro com os seguintes atributos: **marca** (String), **modelo** (String), **ano** (int). Adicione um método **exibirInfo()** que imprime as informações do carro. No método main, crie duas instâncias de Carro e chame o método **exibirInfo()** para cada uma.
2. Crie uma classe Contador que possui: Um atributo estático **totalObjetos** para contar quantos objetos foram criados; Um construtor que incremente esse contador sempre que um novo objeto for instanciado; Um método estático **mostrarTotal()** que exibe o total de objetos criados. No main, crie alguns objetos e depois exiba o total utilizando o método estático.
3. Crie uma `ArrayList<String>` para armazenar nomes de alunos. Adicione pelo menos 5 nomes. Use um Iterator para percorrer e imprimir cada nome da lista. Em seguida, remova um nome e exiba novamente a lista atualizada.