Significado

Irei colocar aqui o significado e a forma mais correta de cada código em JAVA , do básico desta linguagem :

1. Em Java, `public` é um modificador de acesso que especifica que um método, classe ou variável pode ser acessado por qualquer outra classe. Aqui está uma breve explicação sobre seu uso:
   public double peso;
   public double altura;
2. Int. :

é uma palavra-chave que representa o tipo de dado inteiro. Como (-1 , 0 ,1) .

```
int minhaVariavel = 10;
```

Neste exemplo, `minhaVariavel` é uma variável do tipo `int` que armazena o valor inteiro `10`.

3. Scanner :

A classe `Scanner` é uma parte da biblioteca de entrada do Java que permite que os programadores leiam entrada de vários tipos de dados a partir de várias fontes, como o console, arquivos, etc.

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Cria um objeto Scanner para ler entrada do console
        Scanner scanner = new Scanner(System.in);

        // Prompt para o usuário
        System.out.print("Digite um número inteiro: ");

        // Lê um número inteiro fornecido pelo usuário
        int numero = scanner.nextInt();

        // Exibe o número fornecido pelo usuário
```

```
        System.out.println("Você digitou: " + numero);

        // Fecha o scanner após o uso
        scanner.close();
    }
}
```

Neste exemplo, usamos `Scanner` para ler um número inteiro digitado pelo usuário no console. A chamada ao método `nextInt()` lê o próximo token de entrada como um número inteiro. Após a leitura, fechamos o `Scanner` usando o método `close()` para liberar recursos.

4. Main :

```
public class Main {
    public static void main(String[] args) {
        // Código aqui
    }
}
```

Explicando cada parte:

- `public`: É um modificador de acesso, indicando que o método `main` é acessível de qualquer outra classe.
- `static`: Indica que o método `main` pertence à classe em si, e não a uma instância específica da classe.
- `void`: Indica que o método `main` não retorna nenhum valor.
- `main`: É o nome do método. Em Java, o método `main` é o ponto de entrada para o programa.
- `String[] args`: É um parâmetro do método `main`, que é uma matriz de strings. Ele permite que você passe argumentos de linha de comando para o seu programa quando ele é executado.

5. String :

é uma sequência de caracteres. Elas são usadas para armazenar texto, como palavras, frases ou qualquer outra sequência de caracteres.

A classe `String` em Java é usada para manipular strings. Ela fornece vários métodos para realizar operações comuns em strings, como concatenação,

comparação, divisão, substituição, entre outras.

Aqui estão alguns exemplos de operações comuns que você pode realizar com strings em Java:

5.1 **Criar uma String**:

```
String minhaString = "Olá, mundo!";
```

5.2 **Concatenar Strings**:

```
String str1 = "Olá";
String str2 = " mundo!";
String mensagem = str1 + str2;
System.out.println(mensagem); // Saída: Olá mundo!
```

5.3**Converter uma String para Maiúsculas ou Minúsculas**:

```
String minhaString = "Hello";
String maiusculas = minhaString.toUpperCase();
String minusculas = minhaString.toLowerCase();
```

Esses são apenas alguns exemplos das operações básicas que você pode realizar com strings em Java. A classe `String` oferece muitos outros métodos úteis para manipulação de strings.

6. **System :**

A `System` é uma classe fornecida pela biblioteca padrão do Java (`java.lang.System`). Ela fornece acesso a variáveis e métodos relacionados ao ambiente do sistema, entrada e saída padrão, bem como manipulação de propriedades do sistema.

`System.out`: Esta é uma instância de `PrintStream` que é usada para imprimir saída para o console.

7. nextLine :

Parece que você está se referindo ao método `nextLine()` da classe `Scanner` em Java.

O método `nextLine()` é usado para ler a próxima linha de entrada como uma string. Ele captura toda a linha de entrada até encontrar um caractere de nova linha (`'\n'`) e retorna essa linha como uma string, excluindo o caractere de nova linha.

Aqui está um exemplo simples de como usar o método `nextLine()`:

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma linha de texto: ");
        String linha = scanner.nextLine();

        System.out.println("Você digitou: " + linha);

        scanner.close();
    }
}
```

Neste exemplo, o programa solicita ao usuário que digite uma linha de texto e, em seguida, usa o método `nextLine()` para ler essa linha de entrada. O texto digitado pelo usuário é armazenado na variável `linha` como uma string e, em seguida, é exibido de volta para o usuário.

8. Void :

A palavra-chave `void` em Java é usada para indicar que um método não retorna nenhum valor.

Quando um método é declarado com `void` como seu tipo de retorno, isso significa que o método executa determinadas tarefas, mas não produz um valor para ser usado ou manipulado pelo código que o chamou.

Aqui está um exemplo de declaração de um método com tipo de retorno `void`:

```
public void imprimirMensagem() {
    System.out.println("Esta é uma mensagem de exemplo.");
}
```

Neste exemplo, o método `imprimirMensagem()` não retorna nenhum valor. Em vez disso, ele simplesmente imprime uma mensagem na saída padrão.

Por outro lado, se um método precisar retornar um valor, seu tipo de retorno não seria `void`, mas sim o tipo de dado do valor que ele retorna. Por exemplo:

```
public int somar(int a, int b) {
    return a + b;
}
```

Neste caso, o método `somar()` retorna a soma de dois números inteiros e, portanto, tem um tipo de retorno `int`.

9. Double:

Em Java, `double` é um tipo de dado primitivo que representa números de ponto flutuante de precisão dupla. Esse tipo de dado é usado para armazenar números decimais ou números muito grandes ou muito pequenos, pois oferece uma precisão maior do que o tipo `float`.

Aqui está um exemplo de declaração de uma variável do tipo `double`:

```
double numero = 3.14159;
```

Neste exemplo, a variável `numero` é declarada como sendo do tipo `double` e inicializada com o valor `3.14159`.

Você também pode usar a notação científica para representar números muito grandes ou muito pequenos com `double`. Por exemplo:

```
double numeroGrande = 1.2e6; // 1.2 × 10^6 = 1200000
double numeroPequeno = 3.5e-3; // 3.5 × 10^-3 = 0.0035
```

Em operações aritméticas, os números `double` são tratados de maneira apropriada, permitindo operações como adição, subtração, multiplicação e divisão com outros números `double`. No entanto, devido à natureza de ponto flutuante, pode haver pequenos erros de arredondamento ao realizar operações com números `double`.

10. Return :

A palavra-chave `return` em Java é usada para sair de um método e opcionalmente retornar um valor. Ela é usada em métodos que têm um tipo de

retorno diferente de `void` para enviar de volta um valor ao código que chamou o método.

Aqui está um exemplo simples de um método que retorna um valor:

```
public int calcularSoma(int a, int b) {
    int soma = a + b;
    return soma;
}
```

Neste exemplo, o método `calcularSoma` calcula a soma de dois números inteiros `a` e `b` e retorna esse valor como um resultado.

Se um método tem um tipo de retorno `void`, ele não retorna nenhum valor. No entanto, ainda pode usar a palavra-chave `return` para sair do método antes do final, mas nesse caso, é apenas para interromper a execução do método. Por exemplo:

```
public void imprimirMensagem(String mensagem) {
    if (mensagem == null) {
        return; // Sai do método se a mensagem for nula
    }
    System.out.println(mensagem);
}
```

Neste exemplo, se a mensagem fornecida for nula, o método simplesmente sai do método sem imprimir nada. Se a mensagem não for nula, ela é impressa na saída padrão.

11. This :

Em Java, a palavra-chave `this` é uma referência para o objeto atual no contexto em que está sendo usada. Ela pode ser usada em vários contextos para referenciar membros de uma classe, evitar ambiguidades em variáveis locais e passar o próprio objeto como argumento para outro método.

Aqui estão alguns dos usos comuns da palavra-chave `this` em Java:

1. **Referência a Variáveis de Instância**: Dentro de um método de uma classe, você pode usar `this` para se referir às variáveis de instância da própria classe. Isso é útil para distinguir variáveis de instância de variáveis locais que têm o mesmo nome.

```
public class Exemplo {
    private int x;

    public void setX(int x) {
        this.x = x;
         // 'this' refere-se à variável de instância 'x'
    }
}
```

2. **Chamada de Construtores**: Dentro de um construtor, você pode usar `this` para chamar outro construtor da mesma classe. Isso é conhecido como uma chamada de construtor.

```
public class Exemplo {
    private int x;
    private int y;

    public Exemplo() {
        this(0, 0);
        // Chama o outro construtor da mesma classe
    }

    public Exemplo(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

3. **Passagem do Objeto Atual como Argumento**: Você pode passar o próprio objeto como argumento para outro método usando `this`. Isso é útil quando você deseja que o método externo tenha acesso ao objeto atual.

```
public class Exemplo {
    private int x;

    public void metodoA() {
        metodoB(this);
```

```
        // Passa o objeto atual como argumento
    }

    public void metodoB(Exemplo obj) {
        // Faz algo com o objeto passado
    }
}
```

Em resumo, `this` é uma referência ao objeto atual em Java, permitindo uma manipulação eficiente e sem ambiguidades dos membros e métodos de uma classe.

12. If e Else :

Em Java, `if` e `else` são palavras-chave usadas para controle de fluxo condicional. Elas permitem que você execute diferentes blocos de código com base em uma condição.

1. **if**: O `if` é usado para executar um bloco de código se uma condição específica for verdadeira. Se a condição for falsa, o bloco de código não será executado.

```
int idade = 18;

if (idade >= 18) {
    System.out.println("Você é maior de idade.");
}
```

2. **if-else**: O `if-else` é usado para executar um bloco de código se a condição for verdadeira e outro bloco de código se a condição for falsa.

```
int idade = 16;

if (idade >= 18) {
    System.out.println("Você é maior de idade.");
} else {
    System.out.println("Você é menor de idade.");
}
```

3. **if-else if-else**: O `if-else if-else` é usado quando você precisa verificar múltiplas condições. Ele executa o bloco de código associado à primeira condição verdadeira encontrada. Se nenhuma condição for verdadeira e houver um bloco `else`, ele será executado.

```
int nota = 75;

if (nota >= 90) {
    System.out.println("A");
} else if (nota >= 80) {
    System.out.println("B");
} else if (nota >= 70) {
    System.out.println("C");
} else {
    System.out.println("D");
}
```

4. **Operador ternário**: O operador ternário (`? :`) é uma alternativa concisa ao uso de `if-else`. Ele permite que você execute uma operação com base em uma condição em uma única linha.

```
int idade = 20;
String status = (idade >= 18) ? "Maior de idade" : "Menor de
System.out.println(status);
```

Em todos esses casos, o uso de `if` e `else` é fundamental para controlar o fluxo do programa e executar diferentes partes do código com base em condições específicas.