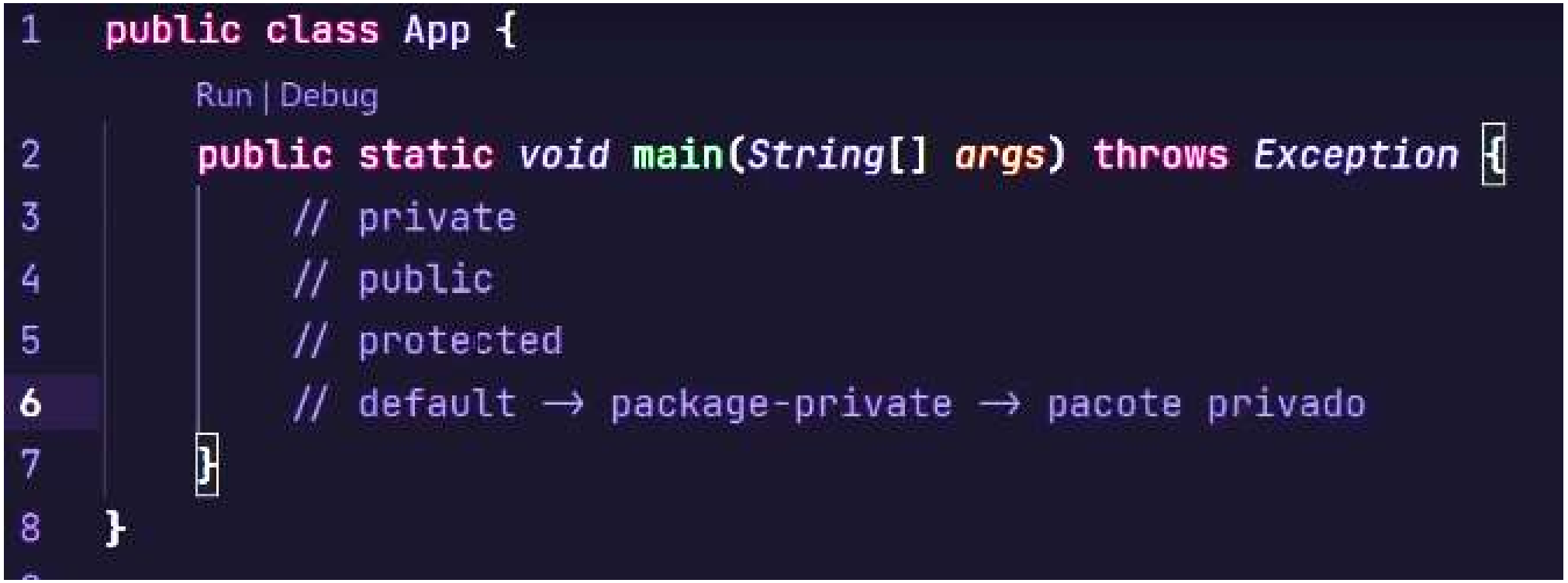
Parte 03

Modificadores de Acesso



Classe

Sintaxe:

```
<modificador de acesso> class <NomeDaClasse> {

    //declaração dos atributos aqui

   //implementação dos métodos aqui

}
```

Exemplo:

```
public class Matematica {


}
```

Modificador de Acesso da Classe

- **`public`**
  - Especifica que a classe pode ser usada por outras classes de outros pacotes;
- (sem o modificador)
  - A não inclusão do modificador torna a classe não visível por outras classes fora do pacote;
  - A classe só poderá ser usada pelas outras classes que estão no mesmo pacote.

Métodos

Sintaxe:

```
<mod. acesso> <mod. método> <tipo retorno> <nomeMetodo>(args) {

    //implementação dos métodos aqui

}
```

Exemplo:

```
public static void someInteiros(int x, int y) {

   int soma = x + y;
   System.out.println("A soma de x e y é : " + soma);

}
```

Modificador de Acesso do Método

- **`public`**
  - Especifica que o método pode ser chamado dentro de outras classes;
- **`private`**
  - Especifica que o método só pode ser chamado dentro da própria classe;
- **`protected`**
  - Especifica que o método só pode ser chamado dentro da própria classe ou por subclasses que o herdam;

Modificador de Método

- **`static`**
  - Indica que o método pode ser chamado sem ter a necessidade de se instanciar a classe que o contém;
  - Métodos estáticos só podem alterar atributos também estáticos;
  - Métodos com esse modificador são chamados de ***métodos de classe***.

Métodos (Retorno de Valores)

Exemplos:

```
public static int someInteiros (int x, int y) {

   return (x + y);

}


public static String cumprimente (String nome) {

   return “Olá “ + nome + “. Tudo bem?”;

}
```

Testando os Modificadores de Acesso

default

```java
public class FestaVip {
    int quantidadeCafe = 30;
    int quantidadeSalgados = 50;

    void beberCafe() {
        quantidadeCafe--;
        System.out.println("Bebeu 1 xícara de café");
    }

    void comerSalgado() {
        quantidadeSalgados--;
        System.out.println("Comeu 5 salgados");
    }
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        FestaVip festa = new FestaVip();
        festa.quantidadeCafe = 100;
        festa.beberCafe();
        System.out.println(festa.quantidadeCafe);
    }
}
```

private

```java
public class FestaVip {
  // <modificador> <tipo> <nome-atributo>
  private int quantidadeCafe = 30;
  private int quantidadeSalgados = 50;


  // <modificador> <retorno-método> <nome-método>
  private void beberCafe() {
    quantidadeCafe--;
    System.out.println("Bebeu 1 xícara de café");
  }

  private void comerSalgado() {
    quantidadeSalgados--;
    System.out.println("Comeu 5 salgados");
  }
}
```

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        FestaVip festa = new FestaVip();
        festa.quantidadeCafe = 100;
        festa.beberCafe();
        System.out.println(festa.quantidadeCafe);
    }
}
```

private - public

```java
public class FestaVip {
    // <modificador> <tipo> <nome-atributo>
    private int quantidadeCafe = 30;
    private int quantidadeSalgados = 50;

    public void entrar() {
        beberCafe();
        comerSalgado();
                ;
    }

    // <modificador> <retorno-método> <nome-método>
    private void beberCafe() {
        quantidadeCafe--;
        System.out.println("Bebeu 1 xícara de café");
```

Chamando um método com modificador **private** por meio de um modificador **public**

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        FestaVip festa = new FestaVip();
        festa.entrar();
    }
}
```

private - public

```java
public class ContaNetflix {
  String idiomaPreferencial;
  String resolucaoTela;

  void entrar() {
    buscarPreferenciasDoUsuario();
    identificarResolucao();
  }

  void buscarPreferenciasDoUsuario() {
    idiomaPreferencial = "PT-BR";
  }

  void identificarResolucao() {
    resolucaoTela = "Full HD";
  }

  void assistirFilme(String nomeFilme) {
    System.out.println("Iniciando o filme " + nomeFilme);
    System.out.println("Carregando o filme na resolução " + resolucaoTela);
    carregarAudioFilme();
  }

  void carregarAudioFilme() {
    if (idiomaPreferencial == "PT-BR" || idiomaPreferencial == "EN-US") {
      System.out.println("Carregando o aúdio em " + idiomaPreferencial);
    } else {
      System.out.println("Carregando o aúdio em EN-US");
    }
  }
}
```

← private

← public

← private

← private

← public

← private

```java
public class App {
    Run | Debug
    public static void main(String[] args) throws Exception {
        ContaNetflix conta = new ContaNetflix();
        conta.entrar();
        conta.assistirFilme("Venom 2");
    }
}
```

Modificador de Método

- **`final`**
  - Especifica que nenhuma subclasse derivada pode alterar ou redefinir este método (ou seja, impossibilita um tipo de polimorfismo: ***sobrescrição de método***);

  - O modificador "final" é usado em programação para indicar que um elemento (geralmente uma classe, método ou variável) não pode ser estendido, sobrescrito ou modificado de qualquer forma. Aqui estão alguns exemplos de como ele é usado:

Modificador de Método

`final`

- **Classe Final:** Uma classe declarada como "final" não pode ser estendida. Ou seja, outras classes não podem herdar dela.
  - `public final class MinhaClasseFinal { // Conteúdo da classe }`

- **Método Final:** Um método declarado como "final" em uma classe não pode ser sobrescrito por subclasses.
  - `public class MinhaClasse { public final void meuMetodoFinal() { // Conteúdo do método } }`

- **Variável Final:** Uma variável declarada como "final" não pode ter seu valor alterado após a atribuição inicial.
  - `public final int minhaVariavelFinal = 10;`

Modificador de Método

- **`abstract`**
  - Indica que o método é abstrato e não tem implementação (corpo).
  - Sua implementação é obrigatória nas subclasses que que o herdam.
  - O modificador "abstract" é usado para declarar classes e métodos abstratos, que são projetados para serem estendidos e implementados por subclasses. Classes abstratas não podem ser instanciadas diretamente, e métodos abstratos não têm implementação na classe pai. Exemplo:

Modificador de Método

abstract

- **Classe Abstrata:** Uma classe abstrata é declarada com o modificador "abstract" e geralmente contém pelo menos um método abstrato.

```
public abstract class MinhaClasseAbstrata { public abstract void meuMetodoAbstrato(); }
```

- **Método Abstrato:** Um método abstrato é declarado sem implementação na classe pai e deve ser implementado em qualquer classe que herde dela.

```
public abstract class MinhaClasseAbstrata {
        public abstract void meuMetodoAbstrato();
}
public class MinhaClasseConcreta extends MinhaClasseAbstrata {
        public void meuMetodoAbstrato() {
        // Implementação do método abstrato
        }
}
```

Tipo de Retorno do Método

- **`void`**
  - Indica que o método não retorna nenhum valor;
- (tipos primitivos)
  - O método pode retornar valores de tipo primitivo, por isso você pode usá-los para especificar o tipo do valor de retorno do método;
- (tipos de classe)
  - O método pode inclusive retorna um objeto inteiro de uma determinada classe.

Instanciação de uma Classe

- Criar objetos a partir de uma classe

```
public class Figura {

}

//aplicativo  que cria um objeto (instância) usando a classe Figura
public class CriacaoDeFiguras {

        public static void main() {

                new Figura();

        }

}
```

Referenciando Objetos

- Para que a instância de uma classe não fique “perdida” na memória e seja manipulada por outra classe é necessário vinculá-la a alguma referência;
- Para isso usamos variáveis que são tipadas pelos próprios nomes das classes;
- Diferentemente dos tipos primitivos, as classes, que especificam os tipos de objetos, são ***tipos por referência***;

Referenciando Objetos

- Para que as instância de uma classe sejam manipuladas

```
public class Figura {

}

public class CriacaoDeFiguras {

        public static void main() {

                Figura objetoFigura;
                objetoFigura = new Figura();

        }

}
```

Métodos de Instância

- Não contêm o modificador de método **`static`**;
- Só podem ser invocados pelas instâncias da classe (objetos);

Métodos de Instância

```
public class Figura {

        public void desenhe() {

                //implementação do método

        }

}

public class CriacaoDeFiguras {

        public static void main() {

                Figura umaFigura = new Figura();
                umaFigura.desenhe();

        }

}
```

Método Construtor

- É o responsável por instanciar a classe;
- Deve ser geralmente público, não ter nenhum modificador de método e não retornar nada (nem conter **`void`**);
- Seu nome deve ser o mesmo da classe;
- Se não for implementado ainda sim ele é implicitamente existente como um método vazio sem argumentos que nada mais faz do que inicializar um objeto.

Método Construtor

```
public class Figura {

        //método construtor
        public Figura() {

                //implementação do método

        }

        public void desenhe() {

                //implementação do método

        }

}
```

Parte 04

Estrutura de Decisão

São utilizadas quando alguma operação (Ex: atribuição, escrever na tela...) está relacionada (depende) da satisfação de uma condição.

```
public class ExemploTesteCond {
    public static void main(String[] args) {
        int x = 20;
        if(x>10){
            System.out.println("O número é maior que 10");
        }
    }
}
```

Estrutura de Decisão IF: “& (AND) (E)”

Operadores relacionados **&& (AND)**

```
public class ExemploTesteCond {
    public static void main(String[] args) {
        int x = 20;
        int y = 5;
        if((x>10) && (y < 10)){
            System.out.println("O número é maior que 10");
        }
    }
}
```

Estrutura de Decisão IF: “|| (OR) (OU)”

Operadores relacionados **|| (OR)**

```
public class ExemploTesteCond {
    public static void main(String[] args) {
        int x = 20;
        int y = 5;
        if((x>10) || (y < 10)){
            System.out.println("O número é maior que 10");
        }
    }
}
```

Estrutura de Decisão IF: “! (NOT) (NÃO)”

Operadores relacionados **! (NOT)**

```
public class ExemploTesteCond {
    public static void main(String[] args) {
        int x = 20;
        int y = 5;
        if(!((x>10) && (y < 10))){
            System.out.println("O número é maior que 10");
        }
    }
}
```

Estrutura de Decisão IF: “ELSE”

```
public class ExemploSeNao {

    public static void main(String[] args) {
        int idade =30;
        if(idade <= 40){
            System.out.println("Jovem");
        }else{
            if(idade <= 60){
                System.out.println("Meia idade");
            }else{
                System.out.println("Idoso");
            }
        }
    }
}
```

Estrutura de Decisão SWITCH

O switch pode ser utilizado somente para testes com valores pontuais (como o valor de x igual 10, igual a 11, igual a 15...). **O switch não pode ser utilizado para testes de faixa de valor (como >10) e pode ser utilizado somente com valores dos tipos int, byte, short ou char**

```
public class TesteSwitch {

    public static void main(String[] args) {
        int x = 1;
        switch(x){
        case 1:
            System.out.println("número pequeno");
        case 10:
            System.out.println("número Grande");

        }
    }
}
```

Estrutura de Decisão SWITCH: “break”

O comando break faz a execução do switch ser interrompida. Caso não o utilizemos para cada case, após a satisfação da condição o switch entrará em todas as opções seguintes

```
public class TesteSwitch {

    public static void main(String[] args) {
        int x = 1;
        switch(x){
        case 1:
            System.out.println("número pequeno");
            break;
        case 10:
            System.out.println("número Grande");
            break;
        }
    }
}
```

Estrutura de Repetição FOR

```
public class ExemploFor {

    public static void main(String[] args) {
        for(int i = 0; i < 10; i++){
            System.out.println("Iteração "+ i);
        }
    }
}
```

Estrutura de Repetição WHILE

```
public class ExemploWhile {

    public static void main(String[] args) {
        int i = 1;
        while(i <= 10){
            System.out.println("Iteração "+ i);
            i++;
        }
    }
}
```