TIPOS DE VARIÁVEIS

Variáveis locais

Existem apenas dentro dos métodos ou dentro dos blocos { } onde são declaradas. São variáveis que têm uma existência curta e uma funcionalidade temporária, pois deixam de ser reconhecidas quando o método ou bloco onde estão definidas termina.

Variáveis de objeto

São declaradas ao nível dos objetos e permitem armazenar os atributos de determinado objeto. Por defeito uma variável é considerada como de objeto.
A sua declaração é efetuada dentro da própria classe em vez de o ser dentro de qualquer método, seja ou não o *main*.
Estas variáveis são conhecidas por todos os objetos que sejam concretizados na classe e podem ser utilizadas por todos eles.

```
class teste {
int x,y;
float a;
public static void main(String args[]){
….
}
```

Variáveis de classe

Quando definimos uma classe e criamos vários objetos dela, já sabemos que cada objeto irá ser uma cópia fiel da classe, porém com suas próprias variáveis e métodos em lugares distintos da memória.
Quando definimos variáveis com a palavra *static* numa classe esta terá um comportamento especial: será a mesma para todos os objetos daquela classe.
Ou seja, não haverá um tipo dela em cada objeto. Todos os objetos, ao acederem e modificarem essa variável, acederão à mesma variável, ao mesmo espaço da memória, e a mudança poderá ser vista em todos os objetos. Funciona de modo semelhante a uma variável global.

```
class teste {
static int x,y;
static float a;
public static void main(String args[]){
….
}
```

Variáveis públicas, protegidas e privada

Uma variável para a qual não foi especificada nenhuma destas palavras-chaves é dita ***amigável*** e pode ser acedida por todas as classes que pertencem ao mesmo pacote.
Uma variável que pode ser acedida por qualquer outra classe é dita pública, e é declarada usando-se a palavra-chave ***public***.
Uma variável que pode ser acedida somente por métodos da própria classe é dita privada, e é declarada usando-se a palavra-chave ***private***.

Uma variável que, além de poder ser acedida por todas as classes do mesmo pacote, também pode ser acedida pelas subclasses da classe na qual ela é declarada, é dita protegida e é declarada usando-se a palavra-chave ***protected***.

Exemplo:

```
package algumPacote; // a classe definida neste ficheiro pertence a algumPacote

public class NomeDaClasse
{
   public int w;          // acessível por qualquer classe
   protected int x;       // acessível pelas subclasses de NomeDaClasse
   int y;                 // acessível pelas classes deste pacote
   private int z;         // acessível somente pelos métodos de NomeDaClasse
   ...
}
```

MÉTODOS

- Quando definimos um objeto num programa orientado a objetos, implementamos todo o comportamento desse objeto em um ou mais métodos.

- Um método em Java é equivalente a uma função, subrotina ou procedimento noutras linguagens de programação. Tal como uma função retorna um valor, mas alternativamente podemos dizer que ele retorna vazio (void), sendo, portanto somente um procedimento.

- Não existe em Java o conceito de métodos globais. Todos os métodos devem sempre ser definidos dentro de uma classe, estará entre as chavetas da classe.

Modificadores de Acesso

O Java controla o acesso a **atributos** e **métodos** através do uso dos modificadores de acesso. São eles:

public	É o menos restritivo de todos. Atributos e métodos declarados como public numa classe podem ser acedidos pelos métodos da própria classe, por classes derivadas desta e por qualquer outra classe em qualquer outro pacote;
protected	Atributos e métodos definidos como protected são acessíveis pelos métodos da própria classe e pelas classes derivadas;
private	É o mais restritivo. Atributos e métodos declarados como private só podem ser acedidos pelos métodos da própria classe.
final	É utilizado para bloquear a extensão de classes ou a sobrescrita de métodos. Poe exemplo: public final class Carro ->impede que a classe Carro tenha subclasses public final void buzinar() -> impede que o método buzinar seja redefinido nas subclasses da classe onde está implementado

Quando nenhum modificador é definido (acesso do tipo "package"), os atributos e métodos podem ser acedidos pelos métodos da própria classe, pelas classes derivadas e por qualquer outra classe dentro do mesmo pacote.

ENCAPSULAMENTO – refere-se ao isolamento entre as partes de um programa. O objetivo do encapsulamento é controlar o tipo de acesso às classes, atributos e métodos o que é conseguido através dos modificadores de acesso. É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.

Todos os atributos deverão ser privados - é desejável que os atributos de um objeto só possam ser alterados poe ele mesmo, inviabilizando situações imprevistas

Habitualmente, os métodos de um objeto são public

Sintaxe (método):

```
<qualificadores> <tipo-de-retorno> <nome-do-método> ([lista-de-
parâmetros]){
<bloco-de-comandos>
}
```

<tipo-de-retorno> é um valor ou objeto que é devolvido pelo método após o processamento interno do método, como em uma função matemática, onde se podem passar parâmetros e ter um valor como resposta. O <tipo-de-retorno> declara-se como um tipo de dados ou uma classe.

<qualificadores> estes podem assumir várias formas, neste momento destaca-se:

public static	Permite criar um método que pode ser executado por agentes externos, inclusive independente de criação de uma instância da classe (instanciação).
private static	Como o nome sugere este método só é visível dentro da própria classe onde foi definido e poderá ser executado diretamente sem necessidade de instanciação.

Os métodos **static** ou **métodos da classe** são funções que não dependem de nenhuma variável de instância, quando invocados executam uma função sem a dependência do conteúdo de um objeto ou a execução da instância de uma classe, conseguindo chamar diretamente qualquer método da classe e também manipular alguns campos da classe.

Para aceder a um método:

Um método de uma classe ou de uma instância da classe é acedido pelo operador ponto (".") na forma:

referência.metodo(params).

A referência é um identificador de:

- objeto, se o método não tiver qualificador *static*.
- classe, se o método tiver qualificador *static*.

Em ambos os casos, omitindo-se a palavra *static*, estaremos a obrigar à instanciação de um objeto para então se poder utilizar o método, um método estático é mais oneroso para o sistema, porém é sempre necessário existir um método estático que inicia o processo de execução.**Exemplo 1:** Exemplo chamada de método static

PROGRAMA OPERACIONAL CAPITAL HUMANO POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

```
public class Teste_Metodo_Static {
	public static void main(String[] args) {
		double num1 = 8.5;
		double pi = Math.PI;

		System.out.println("Valor num1 = "+num1);
		System.out.println("Valor PI = "+pi);
		System.out.println("Soma dos valores = "+(num1+pi));
	}
}
```

Exemplo 2: Exemplo chamada de método static (resultado)

```
class Soma{
	public static int resultado(int num1, int num2){
		return (num1 + num2);
	}
}
public class TestaSomaEstatica {

	public static void main(String[] args) {
		System.out.println(Soma.resultado(10,50));
	}
}
```

No exemplo 2 não é instanciado nenhum objeto, apenas é chamada a classe diretamente, que invoca o método *resultado* que recebe dois argumentos do tipo inteiro.

Palavra reservada *this*

Dentro de um método de instância, o objeto sobre o qual está a ser invocado o método pode ser referenciado pelo *this*.

- Não existe a referência *this* em métodos estáticos (static).
- Normalmente, o *this* é usado para passar o objeto, sobre o qual está a ser chamado o método, como argumento para outros métodos.
- O uso do this não é obrigatório a não ser que num método exista uma variável com o mesmo nome de um atributo. Nesse caso usa-se o this para aceder ao atributo e fazer a distinção. O uso do this aumenta a legibilidade do código por clarificar que se está a aceder a um atributo e não a uma varável local.

Possíveis utilizações:

y=this.x; /*variavel x do corrente objeto*/
this.metodo; /*chamar o método do corrente objeto*/
Metodo(this) /* chamar o método passando o objeto corrente como parâmetro */
return this; /* devolver o objeto corrente como resultado da execução do método */

Exemplo 3:

```
public class Conta {
      int numero=4;
      String dono="Ambrósio Raimundo";
      double saldo=1000;
      double limite=5000;
public static int resultado(int num1, int num2){
            return (num1 + num2);
            }
   public  void levanta(double quantidade) {
  double novoSaldo = saldo - quantidade;
  saldo = novoSaldo;
              }
void deposita(double quantidade) {
       this.saldo += quantidade;
  }
public static void main(String[] args) {
      Conta minhaConta1 = new Conta ();
       System.out.println("Cliente: "+minhaConta1.dono+"\nSaldo atual: " +
minhaConta1.saldo);
        //levanta 200
        minhaConta1.levanta(200);
        System.out.println("saldo da conta de "+minhaConta1.dono+" :
"+minhaConta1.saldo);
         // deposita 500
         minhaConta1.deposita(500);
       System.out.println("saldo da conta de "+minhaConta1.dono+" :
"+minhaConta1.saldo);


      System.out.println(Conta.resultado(10,50));
      }
}
```

CONSTRUTORES

Um construtor é um bloco de código que é executado sempre que utilizamos a palavra reservada new. A sua função é criar uma instância da classe (objeto). É como que um método especial. Atribui-se-lhe o mesmo nome da classe.

Quando o construtor não é explicitamente declarado, o compilador insere o construtor default, isto é um construtor sem parâmetros e com o corpo vazio.
A sua utilização é vantajosa pois possibilita a passagem de argumentos para o objeto durante o processo de criação do mesmo, simplificando o código.

Exemplo (código que correspondente à classe Conta e que define um construtor *default*):

```
public class Conta {

                int numero;
                String dono;
              double saldo;
                double limite;

        public  Conta()
        {   //construtor
         }
 }
```

Exemplo (código que correspondente à classe Conta e que define um construtor definido pelo programador)

```
public class Conta {
        int numero;
        String dono;
        double saldo;
        double limite;
public  Conta(int n, String d,double s, double l) //construtor
{
        numero=n;
        dono=d;
        saldo=s;
        limite=l;
}. . .
```

Cada instância da classe Conta (após a definição do construtor anterior):

```
        Conta minhaConta1 = new Conta (1,"Ambrósio Raimundo",500.0,2500.0);
        Conta minhaConta2 = new Conta (2,"Silvino Bartolomeu",5000.0,1000.0);
```

Exercício resolvido:

```
public class Conta {
        int numero;
        String dono;
        double saldo;
        double limite;
        public  Conta(int n, String d,double s, double l)
        {
                numero=n;
                dono=d;
                saldo=s;
                limite=l;
                System.out.println("no construtor saldo da conta de "+d+" : "+s);
         }

        public static int resultado(int num1, int num2){
                return (num1 + num2);
        }

        public  void levanta(double quantidade) {
         double novoSaldo = saldo - quantidade;    //poderia ser this.saldo-quantidade
                saldo = novoSaldo;
        }
        void deposita(double quantidade) {
         this.saldo += quantidade;
        }
```

```
 public static void main(String[] args) {

       Conta minhaConta1 = new Conta (1,"Ambrósio Raimundo",500.0,2500.0);

       System.out.println("Cliente: "+minhaConta1.dono+"\nSaldo atual: " + minhaConta1.saldo);

        minhaConta1.levanta(200);   //levanta 200

        System.out.println("saldo da conta de "+minhaConta1.dono+" : "+minhaConta1.saldo);

        minhaConta1.deposita(500);  // deposita 500
        System.out.println("saldo da conta de "+minhaConta1.dono+" : "+minhaConta1.saldo);

        System.out.println(Conta.resultado(10,50));

       }
}
```

Exercício 1:

a) Cria uma classe chamada Carro.

b) Adiciona os métodos arrancar() e parar() na classe Carro, sendo as suas instruções constituídas simplesmente pela impressão das seguintes frases "Estou a arrancar!!!", "Estou a parar!!!", respetivamente.

Cabeçalho dos métodos: `public void arrancar()…`

c) Acrescenta o método rolar (int duracao) que calcule a distância percorrida dada duração como parâmetro e sendo a velocidade de 60 (Km/h). No método deverá ser impressa a frase:
Percorri …Km e devolvida distancia.

```
public int rolar(int duracao){
int distancia = duracao* 60;
System.out.println("Percorri "+distancia+" Km.");
return distancia;
}
```

d) Cria uma classe que será relativa ao proprietário da viatura e onde se deverá inserir o método *main* para podermos testar a classe Carro.
Deverá ser instanciada uma variável do tipo Carro para calcularmos a distância percorrida em dois momentos do andamento do veículo (1h e 3h), sendo o 2.º contabilizado a partir do fim do 1.º.

```
public class ProprietarioCarro {
  public static void main(String[] args) {
  // variavel para guardar a distância total percorrida
  int distanciaTotal = 0;
  // instância da classe Carro para efetuar teste
  Carro meuCarro = new Carro();
  // Chamadas dos métodos sobre a variável  meuCarro.
  meuCarro.arrancar();
  distanciaTotal = distanciaTotal + meuCarro.rolar(1);
  distanciaTotal = distanciaTotal + meuCarro.rolar(3);
  meuCarro.parar();
  // Imprimir no ecrã a distancia total percorrida
  System.out.println("Distância total percorrida: "+distanciaTotal
   +" .");
 }
}
```

e) Cria a classe CarroJamesBond , subclasse da classe Carro : será necessário criar a palavra chave "extends ". Nesta classe, no método rolar, a velocidade passará para os 180 Km/h.

```
public class CarroJamesBond extends Carro {
  public int rolar(int duracao) {
   int distancia = duracao * 180;
   System.out.println(":=D YAOUHHHHH !!! Estou a abrir e já percorri
   "+distancia+" Km!!!");
   return distancia;
  }
}
```

f) Cria uma instância de CarroJamesBond, de nome meuCarroJamesBond e usa-a para executar o método rolar, isto é, em vez de meuCarro.rolar(1) e meuCarro.rolar(3) usa meuCarroJamesBond.rolar(1) e meuCarroJamesBond.rolar(3) no cálculo da distância Total.

Bibliografia:

Objetos e classes em Java - Javatpoint
https://www.w3schools.com/java/java_oop.asp
Jesus, C. (2013). Curso Prático de Java.Lisboa:FCA
Coelho, P (2016). Programação em JAVA – Curso Completo. Lisboa: FCA
https://www.devmedia.com.br/metodos-atributos-e-classes-no-java/25404