

PARADIGMAS DE PROGRAMAÇÃO

2023/2024

P.PORTO

ESCOLA
SUPERIOR
DE TECNOLOGIA
E GESTÃO

Aula 03

1. Palavras Reservadas
2. Conceito de Classe
3. Exemplos
4. Modificadores de Acesso
5. Palavras Reservadas Usadas
6. Links Úteis



Palavras Reservadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0



Classes

- ❖ Uma classe é um módulo de *software* que impõe uma dada estrutura
- ❖ Por norma, é uma especificação de um conjunto de características (atributos/propriedades) e/ou de um conjunto de comportamentos (métodos/ funções e/ou procedimentos)
- ❖ Recorrendo a uma classe é possível reutilizar código



- ❖ Reutilizar código definido noutras classes
- ❖ Uma linguagem de programação baseada no paradigma **Object-Oriented**, como é o caso da linguagem Java, possui um conjunto de **APIs – Application Programming Interfaces** – que por sua vez são constituídas por classes
- ❖ É normal as APIs não serem constituídas apenas por classes
- ❖ Os objectos são construídos a partir das classes



```
public class Car {  
  
    static char[] brand = {  
        'f', 'i', 'a', 't'};  
  
    static int passengers = 5;  
    static int doors = 4;  
}
```



```
public class DemoCarOne {  
  
    public static void main(String[] args) {  
        System.out.println("Brand: " + Car.brand[0] +  
            Car.brand[1] + Car.brand[2] + Car.brand[3]);  
        System.out.println("Number of passengers: " +  
            Car.passengers);  
  
        System.out.println("Number of doors: " +  
            Car.doors);  
    }  
}
```



```
public class DemoCarOne {  
  
    public static void main(String[] args) {  
        System.out.println(Car.brand);  
        System.out.println("Number of passengers: " +  
            Car.passengers);  
  
        System.out.println("Number of doors: " +  
            Car.doors);  
    }  
}
```



```
class DemoCarTwo {  
    static Car bmw, volvo;  
  
    public static void main(String[] args) {  
  
        System.out.print("Brand: ");  
        System.out.println(bmw.brand);  
  
        System.out.print("Brand: ");  
        System.out.println(volvo.brand);  
    }  
}
```




```
public class Person {  
  
    static char[] name = {  
        'j', 'a', 'm', 'e', 's'  
    };  
  
    static int age = 25;  
    static int bi = 11223344;  
}
```



```
public class Dog {  
  
    static char[] name = {  
        'f', 'i', 'd', 'o'  
    };  
    static char[] bark = {  
        'w', 'o', 'o', 'f', '!'  
    };  
    static int age = 6;  
}
```



```
public class DemoAll {  
  
    static Car bmw, volvo;  
    static Person person;  
    static Dog dog;  
  
    public static void main(String[] args) {  
  
        System.out.print("Cars Brands: ");  
        System.out.println(bmw.brand);  
        System.out.println(volvo.brand);  
        System.out.print("Person Name: ");  
        System.out.println(person.name);  
        System.out.print("Dog name: ");  
        System.out.println(dog.name);  
    }  
}
```



```
public class FuelPrice {  
  
    static float petrol = 1.2f;  
    static float diesel = 0.98f;  
}  
  
public class Car {  
  
    static char brand[] = {'f', 'i', 'a', 't'};  
    static int passengers = 5;  
    static int doors = 4;  
  
    //p - petrol, d - diesel  
    static char fuelType = 'p';  
    static float literPerKm = 4.5f;  
    static float costPerKm =  
        literPerKm * FuelPrice.petrol;  
}
```



- ❖ O atributo `costperkm` da classe `Car` foi definido à custa do atributo `petrol` da classe `FuelPrice`.

```
static float costPerKm =  
    literPerKm * FuelPrice.petrol;
```

Car

- passengers = 4
- doors = 5
- fuelType = 'p'
- literPerKm

- costPerKm

litterPerKm
* FuelPrice.petrol

FuelPrice

- petrol = 1.2f
- diesel = 0.98f



```
public class DemoCostPerKm {  
  
    static Car car;  
  
    public static void main(String[] args) {  
  
        System.out.print("Brand: ");  
        System.out.println(car.brand);  
  
        System.out.println("Cost per km:" +  
            car.costPerKm + "€");  
    }  
}
```



Controlo de acesso (ou visibilidade)

- ❖ Em Java podem ser especificados quatro tipos de acesso (ou de visibilidade), dos quais três usando palavras reservadas que se designam por modificadores de acesso, mais especificamente **public, private, protected e nenhum**



- Os modificadores de acesso são palavras reservadas que alteram a visibilidade das entidades (classes, interfaces, atributos, métodos, tipos enumerados, blocos de código) às quais estão associados



Modificador de Acesso: *private*

- ❖ Os membros que possuem o modificador `private` só podem ser acedidos dentro do código da própria classe onde foram assinados
- ❖ O modificador `private` é o que permite uma menor visibilidade



- ❖ O modificador `private` é aplicável a:
 - atributos e métodos
 - classes, interfaces, e tipos enumerados que não sejam *Top Level (Outer)*, isto é, a classes aninhadas, interfaces aninhadas, e tipos enumerados aninhados
 - Aninhado (*Nested*) - Definido dentro de outro



```
public class Car {  
    private char[] brand = {'f', 'i', 'a', 't'}  
    private int passengers = 5;  
    private int doors = 4;  
}
```

- ❖ Definir a Classe `Car` deste modo é inútil!
- ❖ Os membros `brand`, `passengers`, e `doors` possuem acesso privado (`private`) como tal não podem ser acedidos senão a partir de código que seja definido dentro da classe `Car`
- ❖ Para mostrar que os membros `brand`, `passengers`, e `doors` não podem ser acedidos a partir de código externo à classe `Car` os mesmos passarão a membros de classe (ver próximo exemplo)



```
public class Car {  
    private static char[] brand = {  
        'f', 'i', 'a', 't'};  
    private int passengers = 5;  
    private int doors = 4;  
}  
  
public class DemoCar {  
  
    public static void main(String[] args) {  
        System.out.print("Brand: ");  
        System.out.println(Car.brand);  
        System.out.print("Passengers: ");  
        System.out.println(Car.passengers);  
        System.out.print("Doors: ");  
        System.out.println(Car.doors);  
    }  
}
```

A partir de DemoCar não é possível aceder aos membros de Car uma vez que os mesmos possuem acesso privado (private)



Modificador de Acesso: *protected*

- ❖ Os membros que possuem o modificador `protected` só podem ser acedidos dentro do código onde foram assinados ou em entidades que derivam daquela a que os mesmos pertencem
- ❖ A aplicabilidade deste modificador é semelhante à do modificador `private`, o seu estudo será aprofundado quando se abordar o conceito de herança



- ❖ Os membros que **não possuem** um dos modificadores `private`, `protected`, ou `public` possuem visibilidade ao nível da `package`
- ❖ A ausência de modificador é aplicável a qualquer entidade (atributos, métodos, classes, interfaces, e tipos enumerados)
- ❖ A ausência de modificador permite um grau de visibilidade igual ao que o modificador `public` permite mas só para uma dada `package`



```
public class Car {
    public static char[] brand = {
        'f', 'i', 'a', 't'};
    static int passengers = 5;
    private int doors = 4;
}

package different;

public class DemoCar {

    public static void main(String[] args) {
        System.out.print("Brand: ");
        System.out.println(Car.brand);
        // "passengers" can't be reached because each
        // class belongs to a different package
        System.out.print("Passengers: ");
        System.out.println(Car.passengers);
    }
}
```



- ❖ Os membros que **possuem** o modificador `public` podem ser acedidos a partir de qualquer código, independentemente da `package`
- ❖ O modificador `public` é aplicável a qualquer entidade (atributos, métodos, classes, interfaces, e tipos enumerados)
- ❖ O modificador `public` é o que permite uma maior visibilidade



Modificador de Acesso: *static*

- ❖ O modificador `static` altera o membro ao qual é aplicado, esses membros passam a chamar-se **membros de classe**
- ❖ Os membros de classe não necessitam de uma instância (objecto) para serem acedidos, para tal basta o nome da classe



- ❖ **Um membro de classe é partilhado por todas as instâncias de uma dada classe**
- ❖ Os membros que não possuem o modificador static chamam-se **membros de instância** e, como é óbvio, **só podem ser acedidos através de um(a) objecto(instância)**



```
public class Car {  
  
    static char[] brand = {'f', 'i', 'a', 't'};  
static int passengers = 5;  
static int doors = 4;  
}  
  
public class DemoCar {  
  
    public static void main(String[] args) {  
  
        System.out.print("Brand: ");  
        System.out.println(Car.brand);  
        // compile error!  
        // "passengers" is an instance member  
        System.out.println(Car.passengers);  
    }  
}
```



Palavras Reservadas Usadas

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used

** added in 1.2

*** added in 1.4

**** added in 5.0



Links Úteis

- <http://docs.oracle.com/javase/tutorial/java/javaOO/classes.html>