02

Aplique a palavra chave static a métodos e campos

O modificador estático diz que determinado atributo ou método pertence à classe, e não a cada objeto. Com isso, você não precisa de uma instância para acessar o atributo, basta o nome da classe.

```
public class Carro {
    public static int totalDeCarros;
}
```

COPIAR CÓDIGO

E depois, para acessar:

```
Carro.totalDeCarros = 5;
```

COPIAR CÓDIGO

Um método estático é um método da classe, podendo ser chamado sem uma instância:

```
public class Carro{
    private static int totalDeCarros;

    public static int getTotalDeCarros() [
        return totalDeCarros;
    }
```

```
}

int i = Carro.getTotalDeCarros();
```

COPIAR CÓDIGO

O que não podemos fazer é usar um método/atributo de instância de dentro de um método estático:

```
public class Carro{
    private static int totalDeCarros;
    private int peso;

    public static int getPeso() {
        return peso;
    }
}
```

COPIAR CÓDIGO

Esse código não compila, pois peso é um atributo de instância. Se alguém chamar esse método, que valor ele retornaria, já que não estamos trabalhando com nenhuma instância de carro em específico?

Repare que a variável estática pode acessar um método estático, e esse método acessar algo ainda não definido e ter um resultado inesperado à primeira vista:

```
    static int b = getMetodo();
    public static int getMetodo() {
        return a;
    }
    static int a = 15;
```

COPIAR CÓDIGO

O valor de `b` será 0, e não 15, uma vez que a variável `a` ainda não foi inicializada e possui seu valor padrão quando da execução do método `getMetodo`.

Outro caso interessante é que o atributo estático for de um tipo não primitivo, que começa com valores null, pode gerar um comportamento não esperado.

```
static Integer inicial = 10;
static Integer segunda = inicial + 5; // compila

static Integer outra;
static void inicializa() {
   outra = 10;
}
static Integer naoExecuta = outra + 1;
// compila mas dá exception em runtime, NullPointerException
durante a inicializacao dessa linha
```

COPIAR CÓDIGO

Um detalhe importante é que membros estáticos podem ser acessados através de instâncias da classe (além do acesso direto pelo nome da classe).

```
Carro c = new Carro();
int i = c.getTotalDeCarros();
```

COPIAR CÓDIGO

Cuidado com essa sintaxe, que pode levar a acreditar que é um método de instância. É uma sintaxe estranha mas que compila e acessa o método estático normalmente.

Além disso, esteja atento pois, caso uma classe possua um método estático, ela não pode possuir outro método não estático com assinatura que a sobrescreveria (mesmo que em classe mãe/filha):

```
class A {
    static void a() { // não compila
    }
    void a() { // não compila
    }
}

class B {
    static void a() {
    }
}
class C extends B {
    void a() { // não compila
    }
}
```

COPIAR CÓDIGO

Outro ponto importante a tomar nota é que o ::binding:: do método é feito em compilação, portanto, o método invocado não é detectado em tempo de execução. Leve em consideração:

```
class A {
    static void metodo() {
        System.out.println("a");
    }
}
```

```
class B extends A {
    static void metodo() {
        System.out.println("b");
    }
}
```

COPIAR CÓDIGO

Caso o tipo referenciado de uma variável seja `A` em tempo de compilação, o método será o da classe `A`. Se for referenciado como `B`, será o método da classe `B`:

```
        A a= new A();
        a.metodo(); // a

        B b= new B();
        b.metodo(); // b

        A a2 = b;
        a2.metodo(); // a
    }
}
```

COPIAR CÓDIGO

A definição de uma variável estática pode invocar métodos e variáveis estáticas:

```
class A {
    static int idade = calculaIdade();
    static int calculaIdade() {
        return 18;
    }
}
```

COPIAR CÓDIGO

A palavra-chave `static` pode ser aplicada a classes aninhadas, mas este tópico não é cobrado nesta primeira certificação.