02

Efeito que ocorre com referências a objetos e a tipos primitivos quando passados a outros métodos

As informações que queremos enviar para um método devem ser passadas como parâmetro. O domínio de como funciona a passagem de parâmetro é fundamental para a prova de certificação.

O requisito para entender passagem de parâmetro no Java é saber como funciona a pilha de execução e o *heap* de objetos.

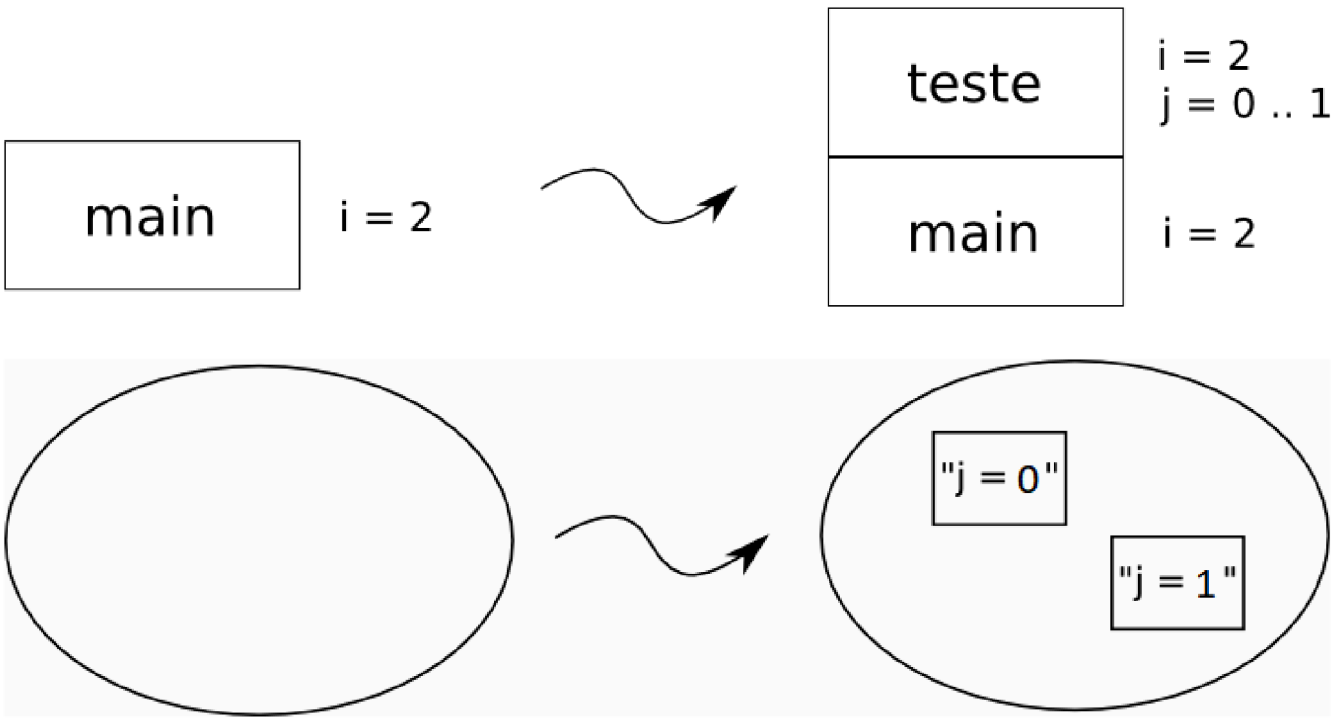
A pilha de execução é o "lugar" onde são empilhados os métodos invocados na mesma ordem em que foram chamados.

O *heap* é o "lugar" onde são guardados os objetos criados durante a execução.

Considere o exemplo a seguir:

```
class Teste {
    public static void main(String[] args) {
        int i = 2;
        teste(i);
    }

    private static void teste(int i) {
        for (int j = 0; j < i; j++) {
            new String("j = " + j);
        }
    }
}
```

COPIAR CÓDIGO



A passagem de parâmetros é feita por cópia de valores. Dessa forma, mudanças nos valores das variáveis definidas na lista de parâmetros de um método não afetam variáveis de outros métodos.

Passagem de parâmetros primitivos

Veja o seguinte código:

```
class Teste {
    public static void main(String[] args) {
        int i = 2;
        teste(i);
        System.out.println(i);
    }

    static void teste(int i) {
        i = 3;
```

```
    }
}
```

COPIAR CÓDIGO

Ao executar a classe `Teste`, será impresso o valor `2`. É necessário perceber que as duas variáveis com o nome `i` estão em métodos diferentes. Há um `i` no `main()` e outro `i` no `teste()`. Alterações em uma das variáveis não afetam o valor da outra.

Passagem de parâmetros de referência

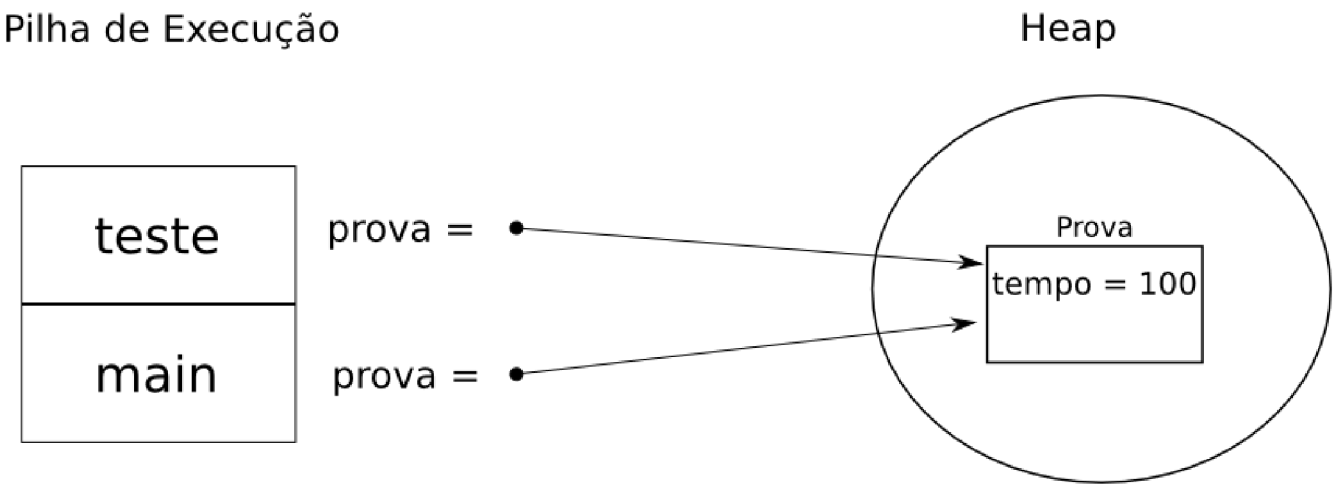
Agora veja esta classe:

```
class Teste {
    public static void main(String[] args) {
        Prova prova = new Prova();
        prova.tempo = 100;
        teste(prova);
        System.out.println(prova.tempo);
    }

    static void teste(Prova prova) {
        prova.tempo = 210;
    }
}

class Prova {
    double tempo;
}
```

COPIAR CÓDIGO

Esse exemplo é bem interessante e causa muita confusão. O que será impresso na saída, ao executar a classe `Teste`, é o valor `210`. Os dois métodos têm variáveis com o mesmo nome (`prova`). Essas variáveis são realmente independentes, ou seja, mudar o valor de uma não afeta o valor da outra.

Por outro lado, como são variáveis não primitivas, elas guardam referências e, neste caso, são referências que apontam para o mesmo objeto. Modificações nesse objeto podem ser executadas através de ambas as referências.



Mas se eu trocar a referência, só estou trocando nesta variável local, e não no objeto referenciado, como no exemplo do `teste2`, em que estamos trocando somente a referência local e não o outro:

```
class Prova {
    int tempo;
}
class TestaReferenciaEPrimitivo {
    public static void main(String[] args) {
        Prova prova = new Prova();
        prova.tempo = 100;
        teste(prova);
        System.out.println(prova.tempo);

        teste2(prova);
```

```
        System.out.println(prova.tempo);



        int i = 2;
        i = teste(i);
        System.out.println(i);
    }
    static void teste2(Prova prova) {
        prova = new Prova();
        prova.tempo = 520;
    }

    static void teste(Prova prova) {
        prova.tempo = 210;
    }

    static int teste(int i) {
        i = 5;
        System.out.println(i);
        return i;
    }
}
```

COPIAR CÓDIGO