02

Declare, instancie, inicialize e use um array uni-dimensional - Array de referências

Em cada posição de um array de tipos não primitivos é guardada uma variável não primitiva. Esse é um fato fundamental.

```
// Declarando e iniciando um array de Prova
Prova[] provas = new Prova[10];
```

COPIAR CÓDIGO

Lembrando que o `new` inicia as variáveis implicitamente e que o valor padrão para variáveis não primitivas é `null`, todas as dez posições do array desse código estão `null` imediatamente após o `new`.

```
// Erro de execução ao tentar aplicar o operador "."
// em  uma referência com valor null.
// NullPointerException
provas[0].tempo = 10;
```

COPIAR CÓDIGO

Para percorrer um array de tipos não primitivos, podemos utilizar um laço:

```
for (int i = 0; i < provas.length; i++){
    provas[i] = new Prova();
    provas[i].tempo = 210;
}
```

```
for (Prova prova : provas){
    System.out.println(prova.tempo);
}
```

COPIAR CÓDIGO

Caso a classe `Prova` seja abstrata, devido ao polimorfismo é possível adicionar filhas de `Prova` nesse array: o polimorfismo funciona normalmente, portanto funciona igualmente para interfaces.

```
class Prova {
}
class ProvaPratica extends Prova {
}
class Test {
    public static void main(String[] args) {
        Prova[] provas = new Prova[2];
        provas[0] = new Prova();
        provas[1] = new ProvaPratica();
    }
}
```

COPIAR CÓDIGO

Uma vez que o array de objetos é sempre baseado em referências, lembre-se que um objeto não será copiado, mas somente sua referência passada:

```
Cliente guilherme = new Cliente();
guilherme.setNome("guilherme");

Cliente[] clientes = new Clientes[10];
clientes[0] = guilherme;
```

```
System.out.println(guilherme.getNome()); // guilherme
System.out.println(clientes[0].getNome()); // guilherme

guilherme.setNome("Silveira");

System.out.println(guilherme.getNome()); // silveira
System.out.println(clientes[0].getNome()); // silveira
```

COPIAR CÓDIGO

Casting de arrays

Não há ::casting:: de arrays de tipo primitivo, portanto não adianta tentar:

```
int[] valores = new int[10];
long[] vals = valores; // não compila
```

COPIAR CÓDIGO

Já no caso de referências, por causa do polimorfismo é possível fazer a atribuição sem casting de um array para outro tipo de array:

```
String[] valores = new String[2];
valores[0] = "Certificação";
valores[1] = "Java";

Object[] vals = valores;
for(Object valor : vals) {
    System.out.println(valor); // Certificação e depois
```

```
Java
    }
```

COPIAR CÓDIGO

E o casting compila normalmente mas, ao executarmos, um array de `Object` não é um array de String e levamos uma `ClassCastException`:

```
Object[] valores = new Object[2];
valores[0] = "Certificação";
valores[1] = "Java";
String[] vals = (String[]) valores;
for(Object valor : vals) {
    System.out.println(valor);
}
```

COPIAR CÓDIGO

Isso pois a classe dos dois é distinta e a classe pai de `array de string` não é um array de objeto, e sim, um `Object` (lembre-se: todo array herda de `Object`):

```
Object[ ] objetos = new Object[ 2 ];
String[ ] strings = new String[ 2 ];
System.out.println(objetos.getClass().getName());
// [ Ljava.lang.Object;
System.out.println(strings.getClass().getName());
// [ Ljava.lang.String;

System.out.println(strings.getClass().getSuperclass());
// java.lang.Object
```

COPIAR CÓDIGO