07

Faça o que eu fiz na aula

Temos um array com correntistas e suas compras:

```
$correntistas_e_compras = [
    "Giovanni",
    "João",
    12,
    "Maria",
    25,
    "Luis",
    "Luisa",
    "12",
    "Rafael",
];
```

Nesse array, temos uma lista de elementos, e precisamos programar uma função que remova um determinado elemento do array.

Para organizarmos os métodos que precisamos, vamos criar uma classe chamada `ArrayUtils`, e nela criar um método estático com o nome `remover`, como a instanciação desse método não será necessária, o faremos como um método estático.

```
<?php

class ArrayUtils
{
    public static function remover(string $elemento, array &$array): void
    {


    }
}
```

O método remover aceita como argumento um elemento e um array, que será passado como referência utilizando o `&`. Lembrando que passar o array como referência modifica o array em si, e não será criada uma cópia no escopo local da função. Como é o próprio array que passamos como argumento que será modificado, a função não retornará nada, por isso escrevemos void no tipo de retorno.

Para removermos um elemento de um array, precisamos primeiramente verificar se ele está realmente dentro desse array.

Para isso, vamos utilizar a função `array_search`, ela vai utilizar como argumento o elemento que estamos procurando, o array e de acordo com a documentação, ela vai nos retornar um tipo `mixed`. O que é isso?

O tipo `mixed` é uma indicação na documentação do PHP que não é especificado corretamente qual o tipo de retorno daquela função, e pode ser mais de um tipo.

Neste caso, a função retorna o booleano `FALSE` se o elemento não for encontrado no array e caso encontre, retorna o número inteiro com a posição no array.

```
class ArrayUtils
{
    public static function remover(string $elemento, array &$array): void
    {
    $posicao = array_search($elemento, $array);

    }
}
```

Agora que temos a posição, utilizamos a função `unset` para removê-la do array:

```
public static function remover(string $elemento, array &$array): void
{
    $posicao = array_search($elemento, $array);
    unset($array[$posicao]);
}
```

Mas no entanto, apesar de tudo parecer bem esse código está extremamente instável, o que acontece se eu colocar para remover um nome que não existe no array?

Ele removeu a primeira posição! e isso não é o que queríamos, que aconteceu?

O PHP possui uma propriedade em seu sistema de tipos chamada Type Juggling, o que isso faz é que ele tenta ao máximo dar um erro e fazer o seu programa parar, se ele encontra um tipo incompatível para uma operação ele tenta fazer a transformação do tipo, também chamada de casting, automaticamente.

Podemos resolver isso usando um if antes do código:

```
public static function remover(string $elemento, array &$array): void
{
    $posicao = array_search($elemento, $array);
    if ($posicao) {
        unset($array[posicao];
    echo var_dump($array);
    } else {
    echo “Elemento não encontrado no array”;
}
```

Mas ainda existe mais um ponto que pode dar erro nesse código, se removermos a primeira posição, o PHP converterá a posição 0 para `false` , e não removerá do array.

Vamos utilizar a função `is_int` dentro da comparação do `if` .

```
if (is_int($posicao)) {
unset($array[$posicao];
    echo var_dump($array);
} else {
    echo “Elemento não encontrado no array”;
}
```

Agora voltando ao arquivo index.php, fazemos o require do arquivo e chamamos o método:

```
require 'ArrayUtils.php';

ArrayUtils::remover("Giovanni", $correntistas);
```

E vemos que foi removido normalmente! Vamos remover agora a string "12", o penúltimo elemento do array.

```
ArrayUtils::remover("12", $correntistas);
```

E vamos agora executar um `var_dump` no array, antes e depois de efetuar a opção de remoção:

```
var_dump($correntistas);
```

O que aconteceu? a string "12" ainda está lá, e... espera... porque o array diminuiu uma posição?

Observem que no primeiro array, antes de removermos a string "12", o array possuía na segunda posição, um número inteiro 12.

O que acontece é que o `array_search` por padrão, usa a comparação de tipagem fraca com os elementos, ele tenta fazer o casting automático se o conteúdo pode ser convertido.

Como no caso a string "12", o PHP converte para o número inteiro 12, e casa com a posição no começo do array, ao invés do final.

É uma má ideia misturar elementos de tipos diferentes em um array, o PHP não faz diferença e possibilita isso, mas este é um dos motivos do porque isso não é recomendado.

```
public static function remover(string $elemento, array &$array): void
```

Para isso, o `array_search` tem uma opção estrita, um booleano como terceiro elemento que deve ser definido como `true`, SEMPRE devemos utilizar o modo estrito quando uma função oferece isso para a gente.

```
public static function remover(string $elemento, array &$array): void
{
    $posicao = array_search($elemento, $array, true);
    if ($posicao) {
        unset($array[$posicao];
    echo var_dump($array);
    } else {
    echo "Elemento não encontrado no array";
    }
}
```

Legal! agora se entrarmos no browser e tentarmos chamar a função novamente, vemos que ele remove a posição correta, pois agora ele não converte os tipos automaticamente.

Mas observe algo estranho. Estamos tipando o argumento como string, mas se digitarmos um número inteiro, o editor avisa que algo está errado, mas isso não impede o código de executar. E pior, ele faz a conversão automática de tipos de inteiro para string! e isso remove o elemento errado do array.

O comportamento correto seria de ele impedir a execução, se o argumento enviado tiver um tipo diferente do da assinatura do método. Como podemos fazer isso?

O PHP oferece para a gente uma configuração chamada `strict_types` que foi feita para corrigir problemas exatamente iguais a esse que tivemos, essa configuração desabilita a conversão automática de tipos de argumento e retorno de funções.

Ela deve ser a primeira coisa a ser digitada depois da abertura da tag do PHP, e ela funciona somente no arquivo em que ela foi declarada, por isso, recomendo que sempre abra a tag PHP e coloque a declaração `strict_types` em baixo, em todo código.

```
<?php declare(strict_types=1);
```

Com isso, podemos executar novamente o código e vemos que ele não executa se passarmos o tipo errado como argumento da função.