Mistake: treating a reference type as a value type

This lesson will show you how to fix a problem of assuming a reference type as a value type.

Suppose that a customer enters a supermarket and buys a chewing gum. The customer was also a diligent collector of empty bottles and happened to cash in 4000 bottles in exchange for a 1000€ voucher. The cashier is checking if there are enough funds to accept the payment. This check is performed by the `canChange` function returning a boolean result:

```
//shopTransaction for chewing gum
var shopTransaction = {
    items: [ { name: 'Astro Mint Chewing Gum' } ],
    price: 1,
    amountPaid: 1000
}

var cashier = {
    units:    [500, 200, 100, 50, 20, 10, 5, 2, 1],
    quantity: [0, 0, 5, 4, 5, 10, 10, 20, 9]
}

function canChange( shopTransaction, cashier )
{    //amount that has to be paid
    var amount = shopTransaction.amountPaid - shopTransaction.price;
  //checking if the amount can be paid after using
  //every quantity of a unit
  for( var i = 0; i < cashier.units.length; ++i )
    {
        var unit = cashier.units[i];
        while ( amount >= unit && cashier.quantity[i] > 0 )
        {
            amount -= unit;
            cashier.quantity[i] -= 1;
        }
    }

    return amount == 0;
}

console.log( canChange( shopTransaction, cashier ) );
console.log( cashier.quantity );
```

Although the function returns the value we expect, there is still a problem. The side effect of executing the `canChange` function is that all the notes disappeared from the cashier. Why?

The problem is that `cashier` is a reference type. All changes to fields inside the cashier are preserved even after exiting the function.

A possible solution is to create a new variable and equate it to `cashier.quantity[i]`. This will copy the value of `cashier.quantity[i]` to a new location in memory as we are dealing with a primitive type.

```
function canChange( shopTransaction, cashier ) {
    var amount = shopTransaction.amountPaid - shopTransaction.price;
    for( var i = 0; i < cashier.units.length; ++i ) {
        var unit = cashier.units[i];
        var currentQuantity = cashier.quantity[i];
        while ( amount >= unit && currentQuantity > 0 ) {
            amount -= unit;
            currentQuantity -= 1;
        }
    }

    return amount == 0;
}
```

In theory, it is also possible to deeply clone all reference types in functions that are not supposed to mutate their values. Deep cloning is easier said than done. We will explore the different types of cloning in the next chapter.