This class represents a percentage value. This simple class gives you three advantages:

- It encapsulates the logic that handles null values and represents them as percentages.
- You always have two decimal places (by default) in your percentages.
- Better types.

An important note: business logic or calculation is not part of a value object. The only exception I make is basic formatting.

By better types, I mean methods like this:

```
private function averageClickRate(int $total): Percent
{
  return Percent::from(
    SentMail::whereClicked()->count() / $total
  );
}
```

You take a float value and make it a first-class citizen using a Percent value object. You don't have to worry about anymore if a method in your app returns a formatted string or a float number. Every percentage value can be expressed as a Percent object from now on. So you know, it contains the float number and the formatted string value.

The original definition of a value object states two more things:

- It's immutable. You have no setters and only read-only properties.
- It does not contain an ID or any other property related to the identification. Two value objects are equal only when the values are the same.

What else can be expressed as a value object? Almost anything, to name a few examples:

- Addresses. In an e-commerce application where you have to deal with shipping, it can be beneficial to use objects instead of strings. You can express each part of an address as a property: