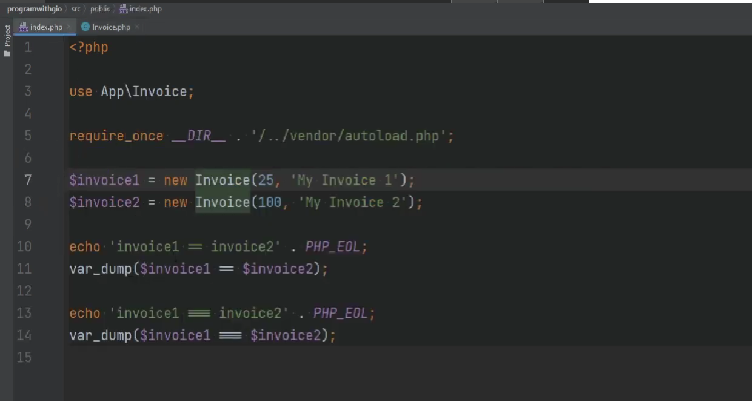
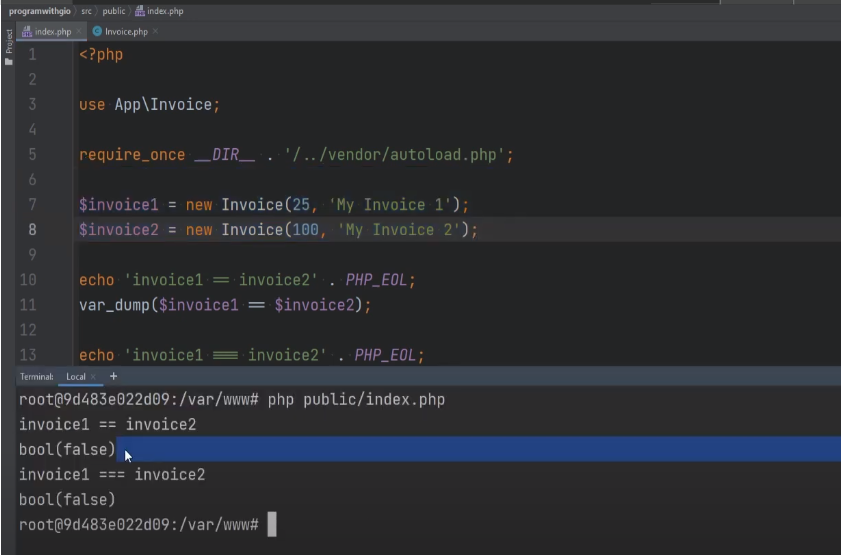
**2.16. Variable storage & object comparison (Lưu trữ biến và so sánh đối tượng)**

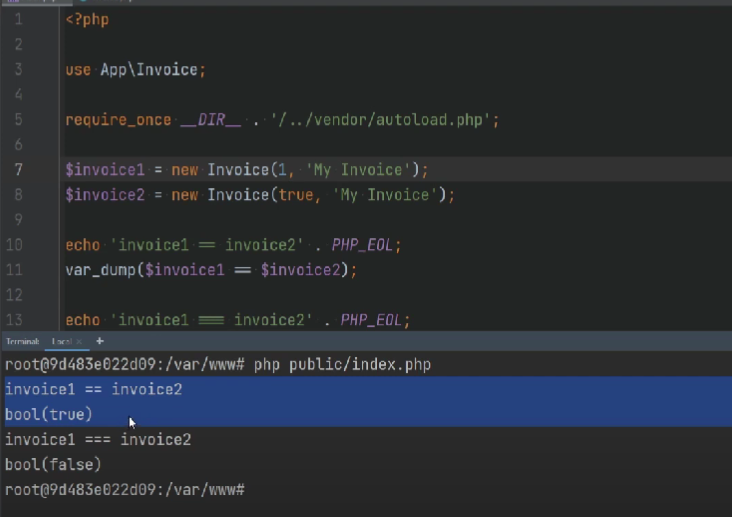
Trong PHP, chúng ta có thể so sánh hai đối tượng bằng nhau bằng cách sử dụng toán tử so sánh "==" hoặc "===". Ví dụ, giả sử chúng ta có hai đối tượng thuộc lớp Invoice. Mỗi đối tượng có hai thuộc tính là số tiền và mô tả. Trong mã PHP, chúng ta có phép so sánh invoice 1 với invoice 2 bằng cách sử dụng toán tử "==" và sau đó sử dụng toán tử "===". Toán tử "==" thực hiện phép so sánh lỏng lẻo, trong đó hai đối tượng sẽ được coi là bằng nhau nếu chúng là các thể hiện của cùng một lớp và có cùng thuộc tính và giá trị. Toán tử "===" thực hiện phép so sánh nghiêm ngặt hơn, nơi cả giá trị và kiểu dữ liệu của hai đối tượng phải giống nhau.



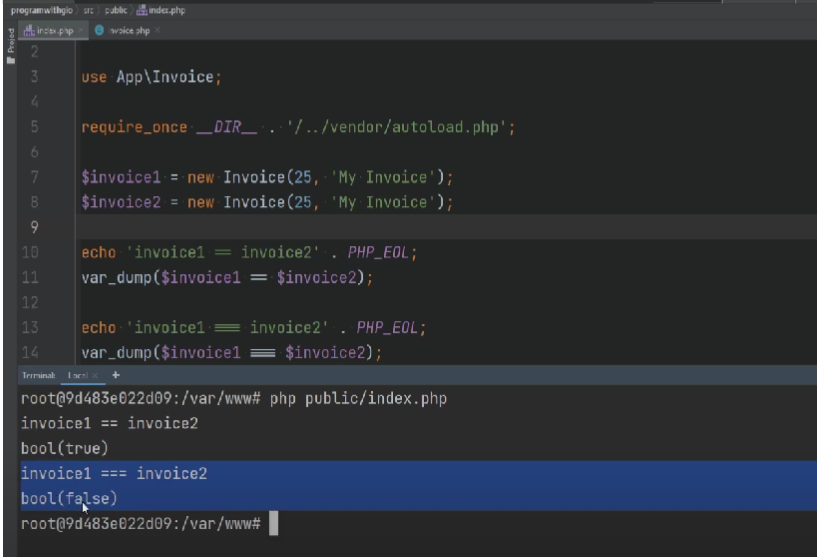
Trong ví dụ, hai đối tượng có các giá trị thuộc tính khác nhau, vì vậy phép so sánh bằng toán tử "==" và "===", cả hai đều sẽ trả về kết quả sai. Một điều cần lưu ý với so sánh dấu bằng kép là nó thực hiện so sánh các giá trị thuộc tính bằng cách sử dụng so sánh lỏng lẻo, ý tôi là giả sử thay vì 25, chúng tôi đã chuyển vào một và thay vì 100



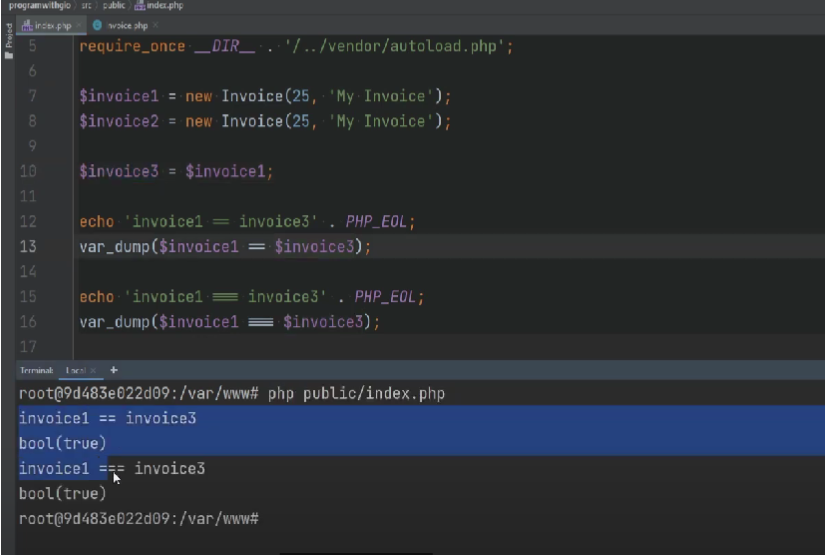
Bây giờ, trong invoice1 và invoice2 tôi thay đổi giá trị lần lượt là số 1 kiểu số nguyên và giá trị true thuộc kiểu boolean và chạy chương trình



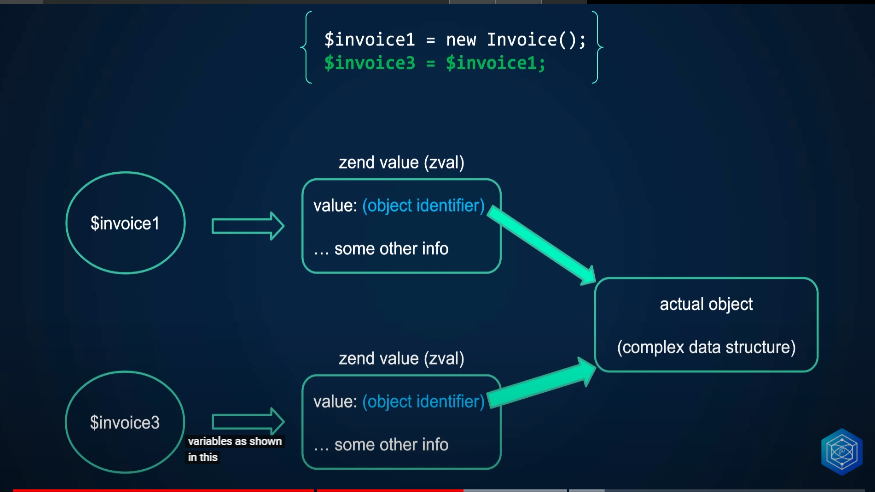
Trong ví dụ này, chúng ta có hai đối tượng có các giá trị thuộc tính khác nhau. Khi chúng ta thực hiện phép so sánh bằng dấu bằng kép (==), PHP sẽ cố gắng tự động chuyển đổi kiểu dữ liệu nếu cần, ngay cả khi chúng ta không kích hoạt chế độ chuyển đổi kiểu tường minh. Vì vậy, mặc dù chúng tôi đang gợi ý loại số nguyên bằng cách sử dụng 25 và true, PHP vẫn sẽ cố gắng tự động chuyển đổi chúng để thực hiện so sánh theo cách của nó. Nói cách khác, dấu bằng kép ở đây thực hiện so sánh kiểu dữ liệu, và khi chúng ta so sánh một số với một boolean theo cách này, nó sẽ trả về true, bởi vì PHP tự động chuyển đổi giữa các kiểu dữ liệu. Vì vậy, nếu tôi chạy mã ngay bây giờ, nó sẽ vẫn trả về true, mặc dù 1 và true không hoàn toàn giống nhau. Vậy, để thực hiện một so sánh nghiêm ngặt hơn, nơi cả giá trị và kiểu dữ liệu phải giống nhau, hãy thay đổi giá trị này thành 25 và cũng đặt giá trị này thành 25.



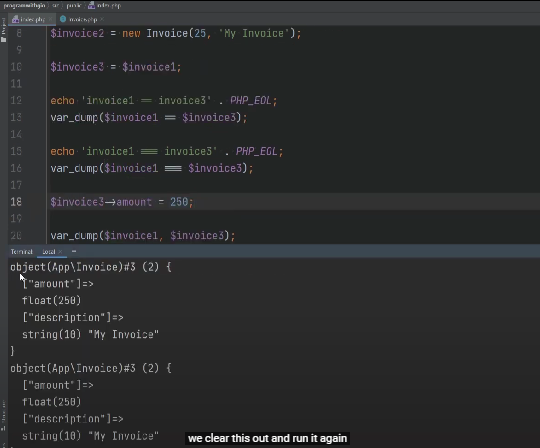
Và bây giờ chúng ta có hai đối tượng khác nhau nhưng chúng có cùng giá trị thuộc tính nếu tôi chạy mã giống như trước khi chúng ta nhận được giá trị true đối với lần so sánh đầu tiên và chúng ta sẽ nhận được giá trị sai cho lần so sánh thứ hai, vậy tại sao chúng ta lại nhận được giá trị sai trong lần so sánh thứ hai, lý do là vì invoice 1 và invoice 2 là hai đối tượng khác nhau mà chúng đang trỏ đến hai phần khác nhau trong bộ nhớ chúng không trỏ đến cùng một đối tượng trong bộ nhớ. Hãy xem ví dụ tiếp theo giả sử chúng ta có invoice3 bằng với invoice1 bây giờ hãy thay đổi so sánh này để so sánh invoice1 với invoice3 thay vì invoice2 và xem những gì chúng tôi nhận được bây giờ chỉ đơn giản là so sánh đối tượng của invoice1 với đối tượng của invoice3. Hãy làm rõ điều này khi chạy mã và chúng tôi đều đúng cho cả hai trường hợp



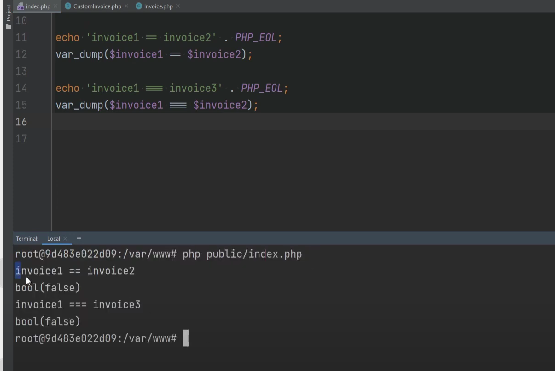
Vì invoice3 thực sự là cùng một đối tượng với invoice mà nó chỉ trỏ đến các đối tượng tương tự mà invoice một trỏ đến trong bộ nhớ invoice1 và invoice3, biến ở đây chỉ đơn giản là các con trỏ trỏ đến một vị trí nào đó trong bộ nhớ chứa đối tượng đó và do đó khi chúng ta gán invoice1 cho invoice3, chúng ta thực sự không sao chép hoặc sao chép hoặc sao chép đối tượng này, chúng ta chỉ đơn giản là tạo một bản sao của một con trỏ sẽ trỏ đến cùng một vị trí trong bộ nhớ chứa đối tượng đó. Vì vậy khi một biến được tạo trong php, php sẽ lưu tên biến thực tế tách biệt với giá trị của nó, tên biến được lưu vào một thứ gọi là bảng ký hiệu nên về cơ bản một biến chỉ là một ký hiệu và mọi biến có một mục trong bảng ký hiệu, mỗi mục trong bảng ký hiệu trỏ đến một loại vùng chứa hoặc cấu trúc dữ liệu nào đó chứa giá trị của biến để bạn có thể coi biến đó chỉ là một con trỏ trỏ tới một loại cấu trúc dữ liệu nào đó vùng chứa này hoặc cấu trúc dữ liệu nơi giá trị được lưu trữ được gọi là "a-z val" và là một phần của cấu trúc ngôn ngữ C. Nó lưu trữ thông tin về biến như loại dữ liệu và giá trị của biến.



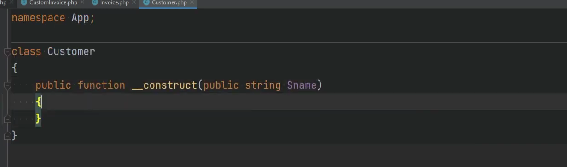
Cách các đối tượng được lưu trữ trong vùng chứa zval này hơi khác so với cách đơn giản khác các kiểu như số nguyên là các giá trị được lưu trữ đối với các kiểu đơn giản như số nguyên được lưu trữ trực tiếp trong bộ chứa zval nhưng đối với các đối tượng, nó chỉ lưu trữ một mã định danh đối tượng mà lại chỉ là một con trỏ tới một cấu trúc dữ liệu phức tạp khác hoặc một kho đối tượng chứa đối tượng thực tế bây giờ khi chúng ta gán invoice1 cho invoice3, chúng tôi lại tạo một biến invoice3 mới được thêm vào bảng ký hiệu và thùng chứa zval được sao chép đơn giản từ invoice1 và invoice3 giờ đây sẽ trỏ đến thùng chứa zval mới đó. chứa cùng một mã định danh đối tượng trong giá trị nên chúng thực sự trỏ đến cùng một vị trí nơi đối tượng thực được lưu trữ, đây là lý do tại sao so sánh ba lần bằng nhau được thực hiện vì đó là cùng một đối tượng mặc dù có các biến khác nhau như được hiển thị trong sơ đồ này, chúng vẫn trỏ đến cùng một đối tượng, chúng tôi có thể chứng minh thêm điều này và chỉ cần var kết xuất cả invoice1 và invoice3 rồi chạy mã và chúng ta thấy rằng chúng thực sự là cùng một đối tượng trên thực tế nếu chúng ta cố gắng thay đổi thuộc tính trên invoice3, giả sử chúng tôi thay đổi số tiền đến mức nào đó như 250,

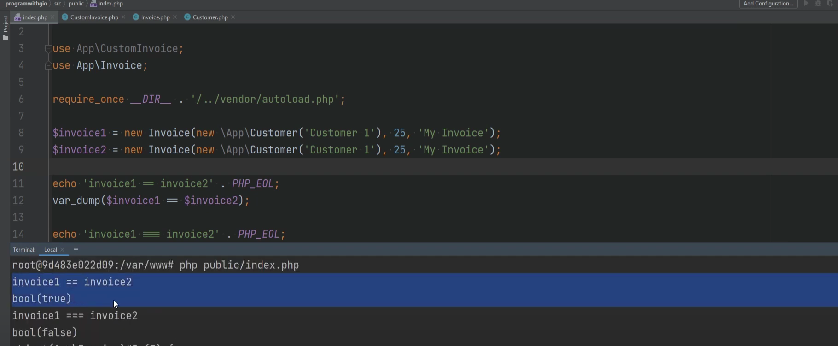


Điều này cũng sẽ ảnh hưởng đến invoice1, vì vậy nếu chúng ta xóa phần này và chạy lại, chúng ta sẽ thấy rằng cả invoice1 và invoice3 đều có thuộc tính được đặt thành 250. Và một lần nữa, điều này là do khi chúng ta thiết lập trong sơ đồ cả hai invoice1 và invoice3 đang trỏ đến cùng một đối tượng trong bộ nhớ nên việc thực hiện các thay đổi sẽ ảnh hưởng đến tất cả các biến trỏ đến cùng một đối tượng, đây là lý do tại sao đôi khi người ta nói và tin rằng các đối tượng theo mặc định được truyền theo tham chiếu, điều này thực tế không đúng. Hãy quay lại mã, hãy tạo một lớp khác gọi là invoice tùy chỉnh và tạo một đối tượng của lớp đó để thay vì có invoice mới cho invoice2 ở đây chúng ta sẽ có invoice tùy chỉnh mới và hãy tạo lớp đó và chúng ta có thể chỉ cần mở rộng invoice thông thường để bằng cách đó chúng ta sử dụng lại cùng một hàm tạo. Bây giờ chúng ta biết rằng cả invoice và invoice tùy chỉnh đều có cùng giá trị thuộc tính vì chúng ta đang tạo hai đối tượng có cùng giá trị, hãy xem điều gì sẽ xảy ra

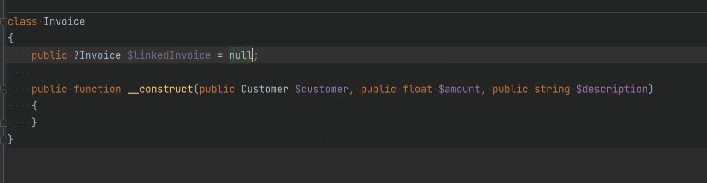


Khi chúng ta so sánh invoice1 với invoice2 bằng cách sử dụng các dấu bằng gấp đôi và gấp ba, hãy loại bỏ điều này, hãy làm rõ mã này và chúng ta thấy rằng nó trả về sai trong cả hai trường hợp và điều này có ý nghĩa, đặc biệt là sau những gì chúng ta thấy trong sơ đồ lần kiểm tra đầu tiên không thành công vì chúng không thuộc cùng một phiên bản invoice1 là một phiên bản của lớp invoice và invoice2 là một phiên bản của lớp invoice tùy chỉnh mặc dù lớp invoice tùy chỉnh hiện mở rộng lớp invoice trong trường hợp so sánh ba lần bằng nhau, điều này cũng không thành công vì nó không phải là cùng một đối tượng, điều này thực sự trỏ đến đối tượng khác và invoice2 trỏ đến đối tượng khác. Lưu ý rằng nếu lớp của bạn chứa một thuộc tính là đối tượng của lớp khác thì việc so sánh được thực hiện theo cách đệ quy, ví dụ như lớp invoice có một thuộc tính được gọi là customer, giả sử rằng chúng tôi đang chấp nhận một đối tượng của lớp customer và chúng tôi xác định thuộc tính đó, hãy tạo lớp này và giả sử rằng lớp này chỉ có một thuộc tính được gọi là tên bây giờ trong index.php, chúng tôi có thể chỉ cần tạo một đối tượng mới đây là custom mới, custom thứ nhất và hãy sao chép customs đó và chúng ta không cần sử dụng invoice tùy chỉnh trong trường hợp này khi chúng ta xóa mã này và chạy mã

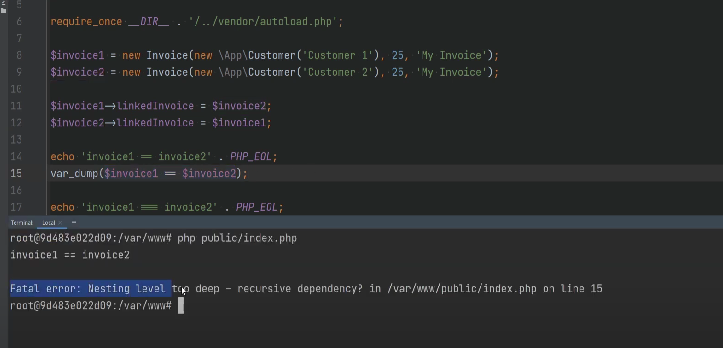




Chúng ta sẽ thấy invoice1 so với invoice2 với dấu bằng kép trả về true nhưng dấu bằng ba trả về sai, nó trả về true cho dấu bằng kép vì tất cả các giá trị thuộc tính đều giống nhau tuy nhiên nếu giá trị thuộc tính của đối tượng lồng nhau khác thì nó sẽ thất bại vì vậy nếu chúng ta thay đổi giá trị này thành ví dụ như customer2 và chúng tôi xóa nó ra và chạy mã, chúng tôi thấy rằng bây giờ invoice1 được so sánh với invoice 2 với invoice 2 có dấu bằng kép trả về sai một điều khác cần lưu ý về điều này là vì nó thực hiện so sánh đệ quy nếu bạn có quan hệ vòng tròn nên nó có thể gây ra lỗi nghiêm trọng khi nói rằng chúng tôi muốn có một thuộc tính trong lớp invoice chứa một đối tượng invoice khác để tạo một số loại liên kết giữa các invoice để chúng ta có thứ gì đó giống như invoice được liên kết invoice công khai và bằng cách đó, chúng ta có thể liên kết các invoice với nhau, chúng ta có thể đặt giá trị này thành null và đặt thành null theo mặc định



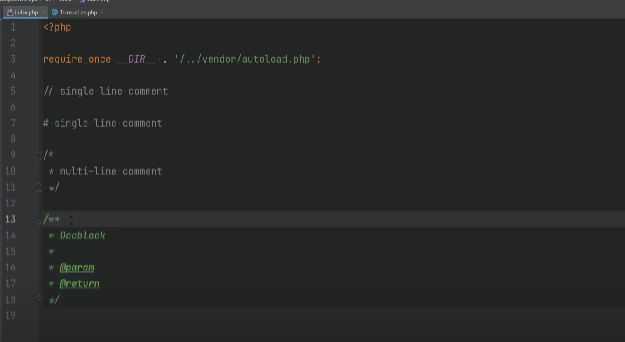
Và bây giờ chúng ta có thể làm điều gì đó giống như invoice được liên kết invoice1 này bằng với invoice2 và invoice được liên kết với invoice2 bằng với invoice1 nên bạn có thể thấy chúng tôi đang tạo mối quan hệ vòng tròn này ở đây khi chúng tôi cố gắng so sánh invoice1 theo các cách để thích điều này và chúng tôi chạy mã mà chúng tôi đang cũng sẽ gặp phải lỗi nghiêm trọng này



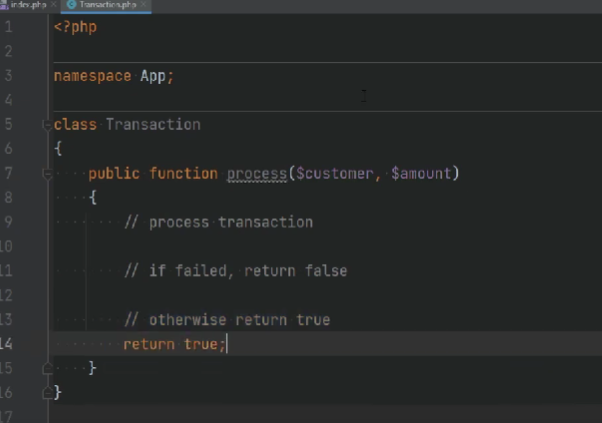
Trước khi chúng ta kết thúc vấn đề này, tôi muốn đề cập là ngoài việc sử dụng các phép kiểm tra đẳng thức như thế này, bạn cũng có thể sử dụng các dấu hiệu lớn hơn hoặc nhỏ hơn để so sánh các đối tượng, nó sẽ dừng và trả về false ở thuộc tính đầu tiên mà không kiểm tra được đây.

**2.17. Adding comments to classes & method (Thêm nhận xét vào lớp và phương thức)**

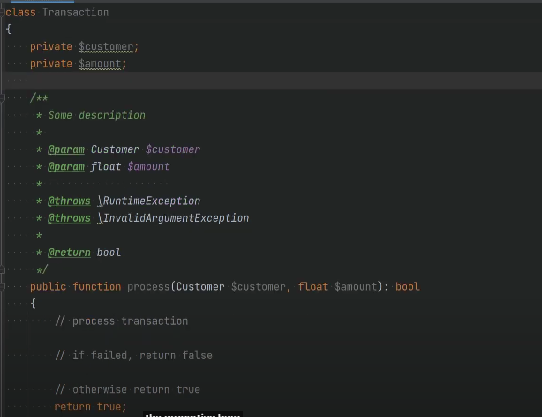
Trong phần này, chúng ta sẽ tìm hiểu cách thêm nhận xét vào các lớp và phương thức trong mã của bạn. Chúng ta đã nói về các loại nhận xét, bao gồm nhận xét một dòng, nhận xét nhiều dòng và khối vịt (Docblock).



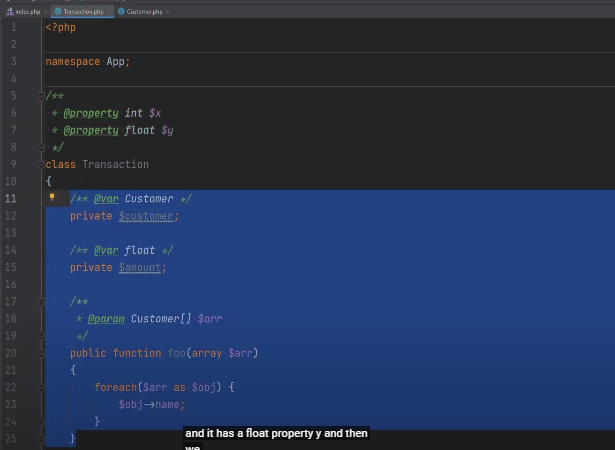
Khối vịt (Docblock) có thể được sử dụng để bổ sung tài liệu cấu trúc cho mã của bạn, giúp tạo tài liệu API tự động và cung cấp thông tin hữu ích cho nhà phát triển khác đang đọc mã của bạn. Ngoài ra còn có một loại khác gọi là khối vịt (Docblock) có thể được thêm vào các hàm, phương thức, lớp, giao diện, biến, thuộc tính và vì vậy, về cú pháp tương tự như nhận xét nhiều dòng, nó chỉ có một dấu hoa thị bổ sung ngay tại đây, khối tài liệu cung cấp tài liệu có cấu trúc tốt hơn cho mã của bạn. Nó có thể được sử dụng để tự động tạo tài liệu API. Các nhà phát triển khác đang đọc mã của bạn có thể sử dụng nó được sử dụng để nâng cao hoặc thay đổi chức năng thông qua chú thích, có thể được ides sử dụng để cung cấp tính năng tự động hoàn thành tốt hơn, v.v., trên các khối tài liệu cũng có thể có văn bản, hãy xem qua một số thẻ mà bạn thường thấy nhất và sử dụng hai thẻ đầu tiên mà chúng ta sẽ xem xét kết quả trả về và văn bản tham số. Thẻ tham số có thể được sử dụng để ghi lại đối số của hàm hoặc phương thức và thẻ trả về có thể được sử dụng để ghi lại kiểu trả về. Các thẻ này đặc biệt hữu ích nếu bạn không gợi ý loại hoặc bạn chỉ muốn cung cấp thêm thông tin về đối số hoặc kiểu trả về.



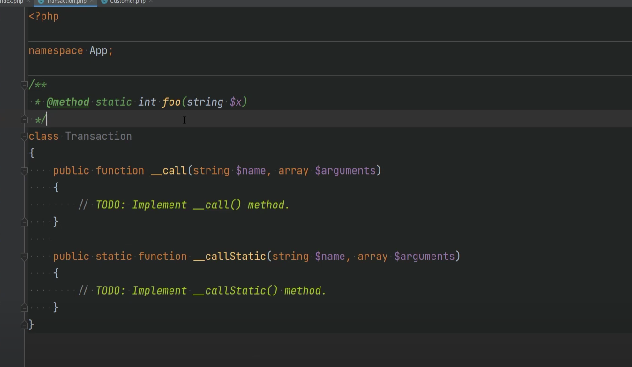
Tôi có một lớp giao dịch (Transaction) ở đây làm ví dụ có một phương thức duy nhất gọi là quy trình chấp nhận hai đối số khách hàng và số tiền và giả sử rằng trong phương thức này, chúng tôi xử lý giao dịch và nếu thất bại, chúng tôi trả về false và nếu thành công, chúng tôi trả về true vì vậy hãy giả sử rằng mọi thứ đều thành công và chỉ cần trả về true ở đây và ở đây, chúng tôi có thể thêm khối tài liệu và như bạn có thể thấy id của tôi tự động hoàn thành khối vịt này sau khi nhấn enter bên trong Các thẻ param chúng ta có thể chỉ định các loại vì chúng ta không gợi ý loại ở đây nên chúng ta có thể nói rằng đối số khách hàng thực sự là loại của lớp khách hàng và số tiền là loại float và sau đó trả về là boolean, bạn cũng có thể tùy ý thêm mô tả ngay sau đó và cũng cung cấp mô tả cho phương thức.



Hiện tại, một số nhà phát triển thích ghi lại mọi thứ. Cá nhân tôi không ghi lại mọi thứ vì bạn có thể chỉ cần gõ gợi ý này ngay, chúng tôi có thể chỉ cần làm khách hàng ở đây và thả nổi ở đây và điều này về cơ bản là như vậy. Điều tương tự như thế này, chúng ta cũng có thể nhập trả về ở đây cho boolean và loại khối tài liệu này trở nên dư thừa trừ khi bạn muốn cung cấp một số thông tin bổ sung về kiểu trả về của đối số hoặc phương thức của chính nó nếu bạn đang mong đợi một đối số đó có thể có nhiều loại hoặc phương thức của bạn có thể trả về nhiều loại thì trước PHP bạn phải sử dụng khối doc và sử dụng ký tự ống để chỉ định nhiều loại, tuy nhiên kể từ PHP 8, giờ đây bạn có thể sử dụng các loại kết hợp để nhập gợi ý nhiều loại thẻ tiếp theo mà chúng ta sắp nói đến là thẻ ném, nó chỉ đơn giản cho biết ngoại lệ hoặc ngoại lệ nào dự kiến sẽ được đưa ra bởi phương thức hoặc hàm này và chúng tôi chưa đề cập đến các ngoại lệ sắp xuất hiện nhưng về cơ bản bạn có thể để chỉ định ngoại lệ nào dự kiến sẽ được ném ra từ phương thức này, do đó bạn sẽ đặt tên của ngoại lệ ở đây, sau đó sao chép nó nếu bạn mong muốn có nhiều loại ngoại lệ được ném ra và đặt nó theo cách đó trong thẻ tiếp theo mà chúng ta đang tìm kiếm.



Điều sắp đề cập là thẻ var dùng để ghi lại loại biến và hằng thuộc tính, giả sử rằng chúng ta muốn có khách hàng và số lượng làm thuộc tính thay vì các đối số được truyền cho phương thức, vì vậy chúng ta sẽ xác định các thuộc tính ở đây và hãy đơn giản Hãy loại bỏ phương thức này ngay bây giờ vì chúng tôi không cần nó và trước PHP 7.4, bạn không thể nhập gợi ý này, vì vậy điều bạn phải làm là sử dụng khối vịt ở đây bằng cách sử dụng thẻ var để chỉ định loại của Thuộc tính này nên chúng tôi sẽ thực hiện khách hàng trong trường hợp này và chúng tôi sẽ thực hiện float trong trường hợp này, tuy nhiên vì PHP bạn cũng có thể trực tiếp trong các thuộc tính, thẻ var này đặc biệt hữu ích khi làm việc với các vòng lặp nếu bạn lặp qua một bộ sưu tập hoặc mảng các đối tượng trong đó mỗi phần tử là một đối tượng của một lớp nào đó, bạn có thể nhập gợi ý rằng bằng cách sử dụng thẻ bar thì các ides như phpstorm sẽ tự động hoàn thành các thuộc tính và phương thức có sẵn trên đối tượng đó



vì vậy, ví dụ: chúng ta có thể có một số loại phương thức foo ở đây nơi chúng ta chấp nhận một mảng và giả sử rằng mỗi phần tử của mảng đó là một đối tượng, chúng ta sẽ làm một cái gì đó như thế này, chúng ta sẽ lặp qua mảng và sau đó có thể chúng ta muốn truy cập các thuộc tính hoặc phương thức trên đối tượng đó để chúng ta có thể gọi một số loại phương thức ở đây bây giờ id sẽ không có thể tự động hoàn thành việc này vì nó không biết đối tượng này thuộc loại gì, bạn có thể nhập gợi ý rằng sử dụng thẻ var ở đây và bạn có thể nói rằng có thể mảng này là một mảng khách hàng nên bạn có thể chỉ định một khách hàng ở đây và bây giờ chúng tôi biết điều đó Phương thức của tôi không tồn tại trên khách hàng nhưng tôi biết rằng đối tượng khách hàng có một tên thuộc tính duy nhất mà tôi đã thêm để chúng tôi có thể truy cập một cách đơn giản rằng chúng tôi thấy id đó đang tự động hoàn thành và chúng tôi thậm chí có thể nhấp vào nó theo cách khác mà bạn có thể làm điều này mà không cần var thẻ là nếu bạn chấp nhận một đối số trong tôi. Nếu bạn chấp nhận một đối số trong một phương thức, bạn có thể chỉ cần thêm khối tài liệu vào phương thức và thay vì kiểu mảng, bạn có thể nói rằng đây là một mảng khách hàng và bạn có thể làm điều đó bằng cách sử dụng loại khách hàng có dấu ngoặc vuông và điều này cho thấy rằng chúng ta có một loạt khách hàng và bây giờ chúng ta có thể loại bỏ điều này và nó vẫn sẽ tự động hoàn thành các thuộc tính và phương thức trên đối tượng đó, cuối cùng, hai thẻ cuối cùng mà tôi muốn đề cập hôm nay là các thẻ thuộc tính và phương thức, những thẻ này cho phép một lớp để biết những phương thức và thuộc tính ma thuật nào có sẵn, hãy nhớ khi chúng ta đề cập đến các phương thức ma thuật để tạo các thuộc tính động và gọi phương thức không tồn tại, những thuộc tính và phương thức đó thực sự không thể được tự động hoàn thành nếu chúng không có sẵn trực tiếp trong lớp ngay với sự trợ giúp trong số các thẻ này, bạn có thể chỉ định những thuộc tính và phương thức nào có sẵn trong lớp này, ví dụ: chúng ta có thể nói rằng lớp này có thuộc tính số nguyên x và nó có thuộc tính float y và sau đó chúng ta giả sử rằng chúng ta đang sử dụng các phương thức getter và setter ở đây để tạo các thuộc tính động này ngay bây giờ khi chúng tôi có đối tượng giao dịch, các id sẽ có thể tự động hoàn thành x và y vì chúng tôi đã thêm thẻ thuộc tính này vào đây ngay bây giờ nếu bạn chỉ muốn gợi ý rằng thuộc tính chỉ dành cho mục đích đọc hoặc thuộc tính chỉ dành cho mục đích viết, nghĩa là nó chỉ đọc hoặc chỉ viết, bạn có thể chỉ cần đọc thuộc tính và viết đúng một ide sẽ chỉ gạch chân cách sử dụng không đúng nhưng nếu bạn muốn có sẵn cho cả đọc và viết thì bạn chỉ cần sử dụng thuộc tính tương tự như thuộc tính bạn cũng có thể có các lệnh gọi phương thức ma thuật, nó có thể có lệnh gọi dấu gạch dưới và lệnh gọi dấu gạch dưới tĩnh cho các phương thức tĩnh và những điều này có thể gây ra một số phương thức không tồn tại. Chúng có thể gọi các phương thức trên các đối tượng khác hoặc những thứ tương tự và chúng thực sự không thể được tự động hoàn thành trên đối tượng này, bạn có thể làm cho nó tự động hoàn thành bằng cách thêm thẻ phương thức vào đây và chỉ định kiểu trả về, vì vậy, giả sử rằng chúng ta có một phương thức trả về số nguyên và phương thức đó được gọi là foo và nó chấp nhận chuỗi làm đối số và bây giờ khi chúng ta sử dụng một đối tượng của lớp này, phương thức 4 sẽ có sẵn để trả về số nguyên và chấp nhận đối số chuỗi x nếu bạn muốn tạo tĩnh này thì bạn sẽ thêm tĩnh ngay ở đầu lưu ý rằng không có thẻ nào trong số này thêm bất kỳ loại xác thực nào mà chúng có thể thêm một số cấp id xác thực trong đó id có thể gạch chân mọi thứ dựa trên thẻ nhưng nó sẽ không ảnh hưởng đến chú thích thực thi mã của bạn về cơ bản là một dạng thẻ khác ngoài việc ghi lại mã, nó còn có thể ảnh hưởng đến cách hoạt động của mã để nó cho phép bạn nhúng siêu dữ liệu bên trong khối tài liệu mà sau đó có thể được xử lý bằng một số công cụ hoặc khung công cụ gói PHP đã giới thiệu một thứ gọi là thuộc tính có mục đích tương tự và sẽ bao gồm các thuộc tính.