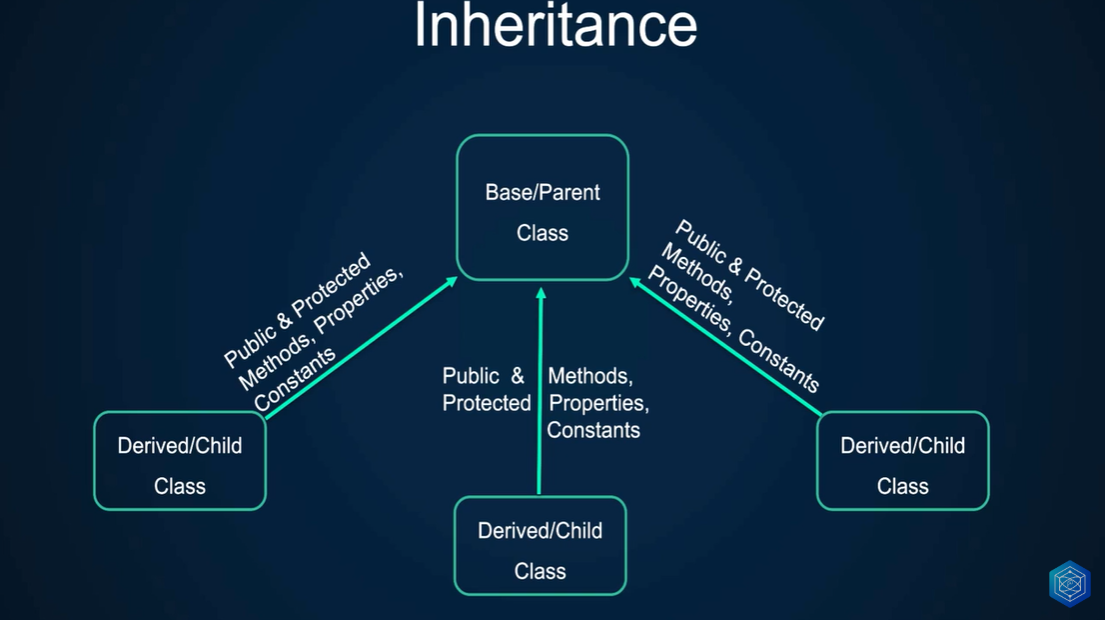
**2.9. Inheritance Explained – Is Inheritace good? Giải thích về tính kế thừa – Kế thừa có tốt không?**

Tính kế thừa là 1 lớp dẫn xuất từ 1 lớp khác cũng được gọi là lớp cha con. Trong đó lớp cha là lớp cơ sở. Lớp con sẽ kế thừa các hằng số và thuộc tính của phương thức công khai (public) và được bảo vệ (protected) mà sau đó có thể được ghi đè. Ngoài việc kế thừa những thứ này trong lớp con, còn có các hằng thuộc tính và phương thức bổ sung.

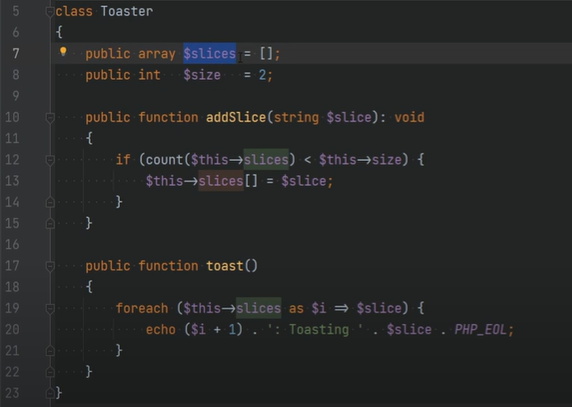


Một số ví dụ:

* Trong mã thực tế:
* Triển khai một chương trình mô phỏng máy nướng bánh mì có thể nướng bánh mì:



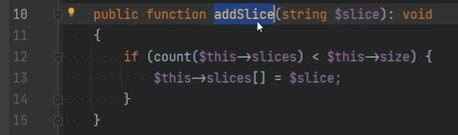
Ở đây có một lớp máy nướng bánh mì **Toaster**.



Nó có 2 thuộc tính chung, một số trong đó là **slices**: các lát lưu trữ, các lát sẽ được nướng và thuộc tính còn lại là **size**: kích thước số lát tối đa mà máy nướng bánh mì này có thể nướng cùng lúc.



Sau đó, có phương thức thêm lát **addSlice**: chỉ cần thêm lát vào mảng lát và nó chỉ thực hiện điều đó nếu chúng ta chưa đạt đến số lát tối đa.



Cuối cùng là phương thức nướng bánh mì **toast**: lặp qua lặp lại mảng lát và in lát bánh mì nướng ra màn hình.

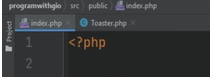


Tại sao sử dụng hằng số ở đây?



Vì kích thước không thực sự thay đổi trong thời gian chạy, máy nướng bánh mì này chỉ có thể nướng 2 miếng cắt lát cùng lúc nên sẽ hợp lý nếu coi cái này là hằng số. Các hằng số và biến **static** ( giá trị của nó được lưu trữ trong suốt quá trình chạy chương trình) được nói rõ trong một video riêng biệt sau khi chuyển sang chủ đề liên kết tĩnh.

Bắt đầu kiểm tra điều này trên index.php.



Tạo đối tượng của lớp Toaster



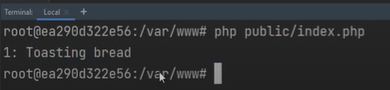
Thêm 1 lát bánh mì.



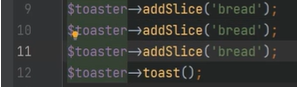
Và gọi phương thức toast.



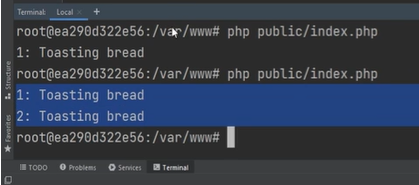
Mở terminal và chạy php public.index.php và chúng ta sẽ thấy bây giờ nó đang nướng bánh mì.



và hãy thêm vài lát nữa vào đây và chạy lại

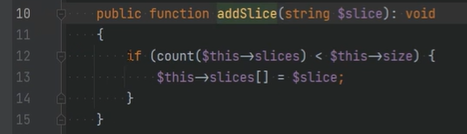


và như bạn có thể thấy, nó chỉ nướng 2 lát mặc dù đã thêm 3 lát ở đây.



Vì số lát tối đa mà Toaster này có thể nướng là 2 lát.

Cách lý tưởng là ném một ngoại lệ vào đây

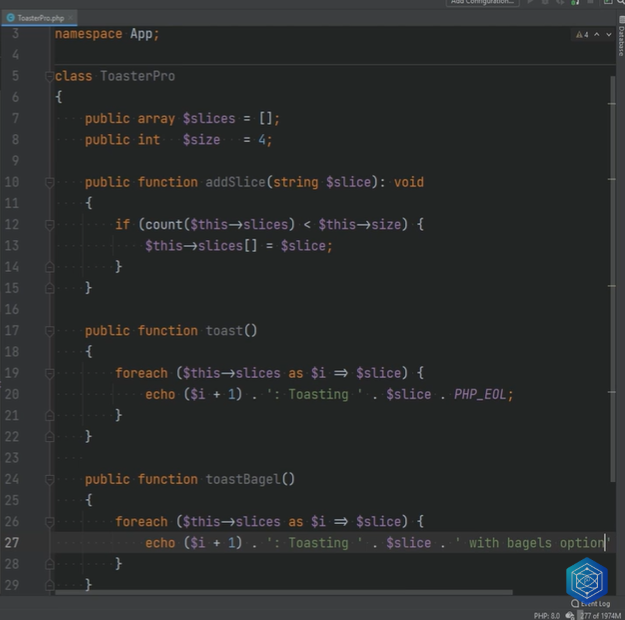


Nếu chúng ta đã đạt đến số lượng lát cắt tối đa nhưng vì chúng ta chưa đề cập đến các ngoại lệ nên tôi quyết định giữ điều này và thực hiện theo cách khác.

Sau này chúng ta muốn triển khai mô phỏng cho một Toaster cao cấp hơn có nhiều chức năng hơn một chút và đơn giản đó là phiên bản chuyên nghiệp của bánh mì nướng ban đầu. Thay vì chúng ta chỉ nướng 2 lát bánh mì cùng lúc, phiên bản chuyên nghiệp có thể nướng 4 lát cùng lúc và ngoài việc có các khả năng nướng cơ bản, nó còn cài đặt sẵn để nướng nó như bánh mì tròn ( bagel).

Vì vậy hãy tạo lớp đó.

Nhân đôi lớp máy nướng bánh mì Toaster và tạo máy nướng bánh mì chuyên nghiệp **ToasterPro** thay đổi size thành 4 để nướng 4 lát cùng lúc. Thêm 1 phương thức để nướng với tùy chọn nướng bánh mì tròn **toastBagel** tương tự như phương pháp toast. Có phương thức thêm lát **addSlice**, giống hệt như phương thức nướng bánh mì thêm lát addSlice ở lớp Toaster. Có phương thức nướng **toast** hoàn toàn giống như phương thức nướng bánh mì toast của lớp Toaster.



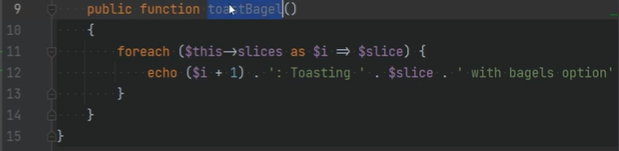
Bởi vì ToasterPro cũng là Toaser. Chú ý mối quan hệ ở đây: ToasterPro là Toaster, nghĩa là nó có thể nướng bánh mì với các cài đặt cơ bản thông thường nhưng nó cũng có thể nướng bánh mì như cài đặt bánh mì tròn bagel và nó có thể nướng 4 lát cùng lúc thay vì 2 lát.

Vì vậy, chúng ta có thể tận dụng sức mạnh của sự kế thừa và kế thừa từ lớp Toaster trong lớp ToasterPro và chúng ta có thể làm điều đó bằng cách sử dụng từ khóa **extends** để chúng ta có thể mở rộng ToasterPro.

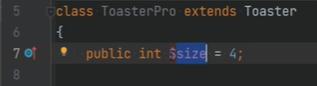


và bây giờ ToasterPro kế thừa tất cả các thuộc tính và hằng số của phương thức public và protected của lớp Toaster.

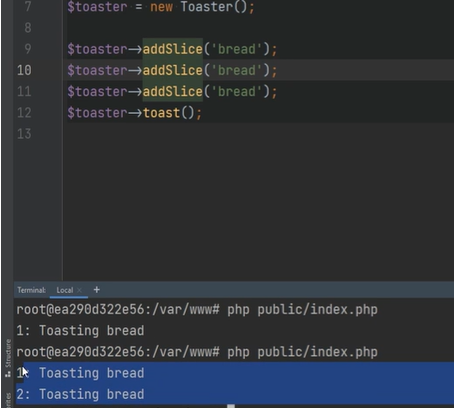
Điều đó có nghĩa là chúng ta có thể loại bỏ phương thức addSlice và phương thức toast. Chúng ta chỉ còn lại phương thức bánh mì tròn nướng **toastBagel**.



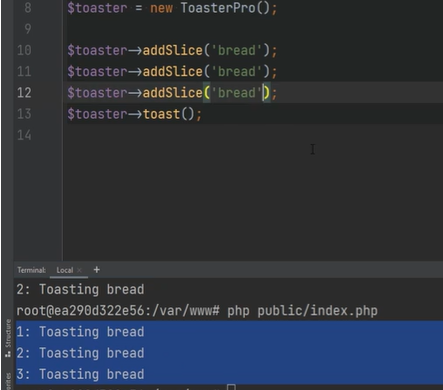
Như đã đề cập trước, bạn có thể ghi đè các hằng số và phương thức thuộc public và protected. Chúng ta sẽ ghi đề thuộc tính size với giá trị là 4.



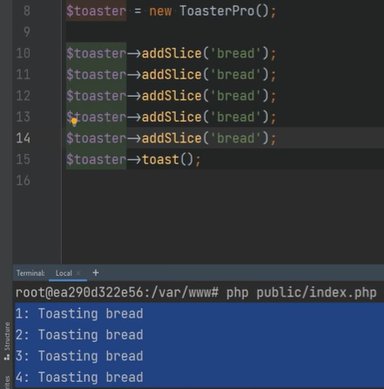
Bởi vì ToasterPro có thể nướng 4 lát cùng lúc, trong khi Toaster chỉ có thể làm 2 lát, nên nếu truy cập index.php và mở terminal thì nó chỉ có thể nướng 2 lát.



Nhưng nếu đổi Toaster() 🡪 ToasterPro() và chạy lại, chúng ta thấy rằng bây giờ nó đang nướng 3 lát.



Có thể thêm vài lát nữa ở đây và chúng ta thấy số lát tối đa là 4.



Vì đó là những gì chúng ta đã cài đặt.

Ngoài việc nướng nó như bánh mì thông thường, chúng ta cũng có thể nướng bánh mì tròn nên chúng ta có thể gọi toastBagel và chạy nó.

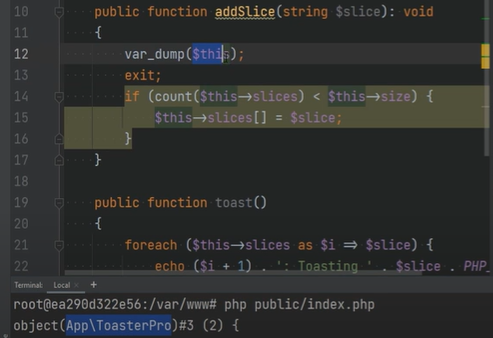


Chúng ta thấy rằng đó là nướng bánh mì với tùy chọn bánh mì tròn cũng có một điều đáng chú ý là: nếu bạn nhớ định nghĩa của biến **this** từ một trong những bài học trước, biến này đề cập đến đối tượng được gọi. Trong trường hợp này khi chúng ta truy cập các thuộc tính hoặc phương thức bằng biến this, điều đó không nhất thiết có nghĩa là nó đang truy cập các thuộc tính đó trên lớp cha mà nó có thể truy cập chúng trên lớp con.

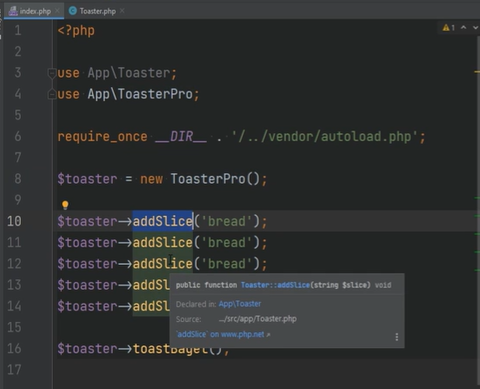
Thay đổi biến này trong lớp cha



và hãy xem những gì chúng ta nhận được, bạn có thể thấy biến này đề cập đến lớp ToasterPro



Nếu chúng ta quay lại index.php, khởi tạo ToasterPro ở đây và chúng ta đang gọi phương thức từ lớp Toaster

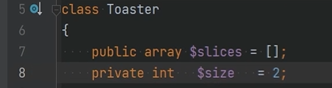


Vì ToasterPro kế thừa hoặc mở rộng từ lớp Toaster, đó là lý do tại sao biến this trong lớp cha hoặc lớp Toaster lại đề cập đến ToasterPro.

Bây giờ chúng ta đã biết những điều cơ bản về kế thừa, hãy nói về một số chi tiết quan trọng khác.

* Một số sơ đồ và một số trường hợp sử dụng tính kế thừa, thời điểm không nên sử dụng tính kế thừa

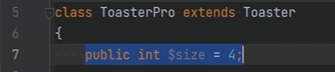
Nếu chúng ta đặt thuộc tính size ở lớp cha thành private và chạy.



Chúng ta thấy rằng bây giờ ToasterPro chỉ nướng 2 lát

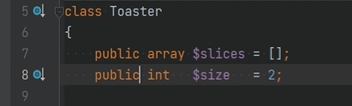


mặc dù đã chỉ định 4 lát ở đây

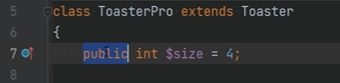


Bởi vì bạn không thể ghi đè lên các thuộc tính private. Các thuộc tính private chỉ thuộc về và tồn tại trong lớp đó ngay cả lớp mở rộng lớp Toaster cũng không thể truy cập các thuộc tính private mà bạn chỉ có thể truy cập và ghi đè các phương thức và hằng số thuộc tính public và protected.

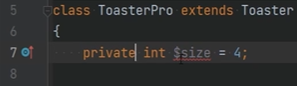
Nếu bạn đặt size ở lớp cha thành public



bạn không thể giảm mức độ truy câp của size nên bạn không thể đặt size ở lớp con thành private,

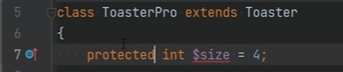


nó sẽ gây ra lỗi cho bạn và nó sẽ báo lỗi và gạch chân lỗi nếu tôi chạy nó.

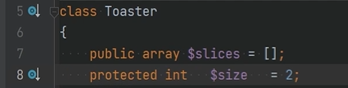


Bạn cũng không thể đặt nó là protected bởi vì protected thấp hơn public.

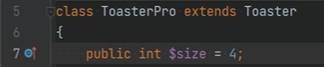
Về cơ bản bạn đang giảm khả năng truy cập hoặc mặt khác nó sẽ không hoạt động nếu bạn đặt cái này thành protected.



bạn có thể ghi đè size ở lớp cha Toaster bằng protected

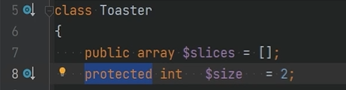


Trong trường hợp này ToasterPro sẽ hoạt động khi chạy nó, mọi thứ đều hoạt động hoặc có thể đặt thành public.

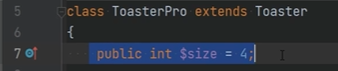


và điều này vẫn sẽ hoạt động theo các quy tắc chính xác tương tự áp dụng cho các phương thức, thuộc tính tĩnh, phương thức và hằng số.

Cũng như bạn biết rằng các thuộc tính public không lý tưởng vì chúng phá vỡ sự đóng gói. Ngay trong trường hợp này, thuộc tính size được đặt protected trong lớp cha Toaster



nhưng lại đang ghi đè ở lớp con bằng khả năng truy cập public,

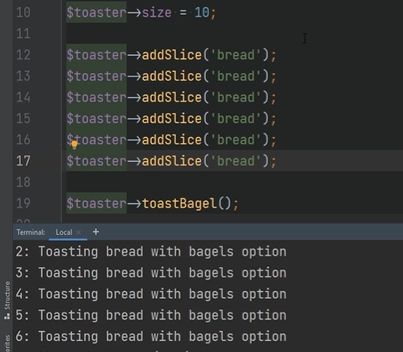


điều đó có nghĩa là chúng ta đang tăng khả năng hiển thị trong lớp con ToasterPro.

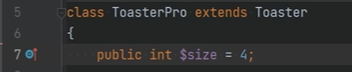
Chúng ta có thể thực hiện tăng size nướng bánh lên 10



ToasterPro chỉ có thể nướng 4 lát cùng lúc nhưng chúng ta đã thay đổi điều đó và nếu thêm nhiều lát hơn và chạy nó, chúng ta có thể nướng nhiều hơn 4 lát.



Một cách khắc phục như bạn biết là đặt size ở lớp ToasterPro thành private.



nhưng sau đó chúng ta không thể ghi đè các thuộc tính.

Một cách khắc phục khác là sửa đổi truy cập protected trong lớp Toaster.

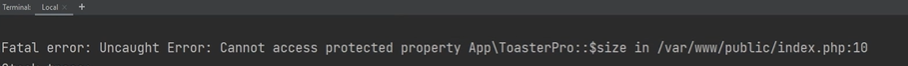


Về cơ bản protected giống như private nhưng nó mở rộng sang các lớp con.

Vì vậy hãy đặt mọi thứ là protected



Và nếu tôi chạy chúng ta sẽ gặp phải lỗi này



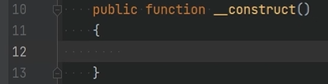
chúng ta không thể truy cập thuộc tính size nữa.

Vì vậy hãy nói về ghi đè phương thức.

Tôi sẽ cho bạn xem một ví dụ về ghi đè các phương thức khác.

Đầu tiên tôi sẽ chỉ cho bạn thấy một ví dụ về ghi đè khác. Như bạn đã nhận thấy, chúng ta không có bất kỳ hàm tạo nào trong lớp Toaster hoặc ToasterPro.

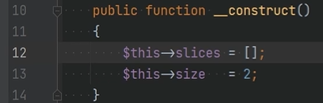
Đó là vì các hàm tạo không bắt buộc, hãy thêm phương thức hàm tạo vào lớp Toaster



và thay vì khởi tạo thuộc tính size thành 2 ở đây ( ở ngoài hàm tạo)

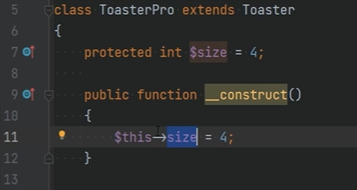


chúng ta có thể chỉ cần làm trong hàm tạo theo cách này:



và loại bỏ phần khởi tạo bên ngoài.

Hãy thực hiện điều tương tự cho ToasterPro, chúng ta có thể chỉ cần ghi đè lên thuộc tính size thành 4 và vì size được đặt protected nên chúng tôi có thể truy cập thuộc tính đó một cách chính xác.



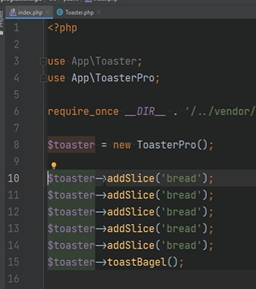
và chúng tôi có thể xóa thuộc tính này ( thuộc tính cũ ở bên ngoài hàm tạo)



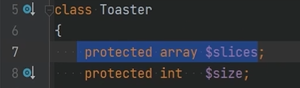
nếu tôi chạy lại nó, chúng tôi gặp phải lỗi lạ.

nói rằng không được truy cập slices trước quá trình khởi tạo.

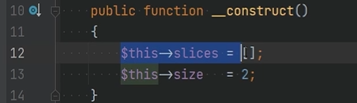
Vậy điều gì đang xảy ra ở đây, hãy đi đến dòng 10 của index.php và gỡ lỗi điều này trên dòng 10, chúng ta đang gọi addSlice.



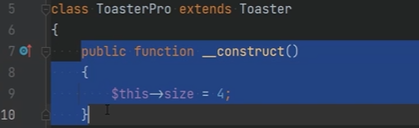
Vậy hãy đi đến addSlice và slices đó nằm trong Toaster và điều này có nghĩa là nó đang truy cập vào thuộc tính slice ở ngay đây



nhưng nó nói rằng thuộc tính này không được khởi tạo nhưng chúng ta đã khởi tạo nó trong hàm tạo rồi.

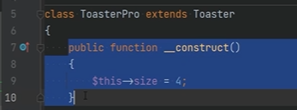


Vậy tại sao nó không hoạt động? Lý do nó không hoạt động là khi bạn ghi đè hàm tạo ở đây ( ở lớp con)



theo mặc định nó sẽ không gọi phương thức construct của lớp cha.

Vì vậy nếu tôi comment phần này là phần ghi đè và tôi chạy lại thì mọi thứ vẫn hoạt động không còn lỗi nữa.



Nhưng khi tôi ghi đè phương thức khởi tạo hoặc bất kỳ phương thức nào theo nghĩa đó thì nó sẽ không gọi phương thức của lớp cha theo mặc định.

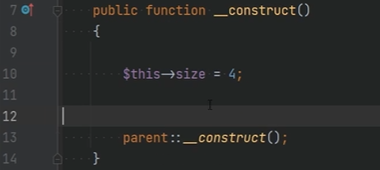
Vì vậy nếu tôi chạy lại, chúng tôi gặp lỗi đó, bạn cần gọi rõ ràng phương thức gốc bằng cách sử dụng từ khóa **parent**, sau đó dùng dấu hai chấm kép là toán tử phân giải phạm vi và sau đó gọi **construct** theo cách này



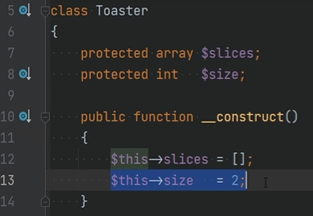
nếu tôi chạy lại thì mọi thứ vẫn hoạt động.

Một điều cần lưu ý ở đây là: bạn cần cẩn thận khi gọi phương thức **parent construct**.

Nếu tôi gọi phương thức parent construct sau khi ghi đè thuộc tính size



thì điều này sẽ gây ra sự cố vì chúng ta đang thiết lập thuộc tính size thành 4. Sau đó chúng ta lại gọi parent construct, mà ở lớp cha đặt thuộc tính size là 2

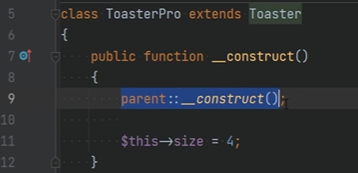


nên về cơ bản nó sẽ đặt lại thuộc tính size là 2.

Nếu tôi chạy code, chúng ta thấy rằng bây giờ nó chỉ nướng 2 lát vì size được đặt thành 2.

Vì vậy hãy cẩn thận khi bạn gọi phương thức parent construct.

Cá nhân tôi luôn đặt nó lên đầu,



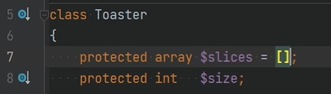
trừ khi tôi có một số logic đặc biệt (Khi tôi cần thực hiện một số nội dung trước khi gọi parent construct như có thể chuẩn bị các đối số cần được thông qua vào hàm tạo gốc, …), nếu tôi chạy lại thì mọi thứ sẽ trở lại bình thường.

Một điều cần lưu ý là: nếu bạn không có hàm tạo trong lớp cha thì bạn không thể gọi parent construct.

Nếu cố gắng làm điều đó chúng ta sẽ gặp lỗi là: không có hàm tạo nào trong lớp Toaster nên tôi cũng cần phải comment điều đó.



Bạn không cần phải có hàm tạo trong lớp cha. Để có hàm tạo trong lớp con và chúng ta không phải khởi tạo mảng slices nên chúng ta cần khởi tạo mảng đó để tránh gặp lỗi.



Tạm thời đặt nó nếu tôi chạy, mọi thứ sẽ trở lại bình thường.

\*\*\*\*

Bạn không cần phải ghi nhớ bất kỳ điều nào trong số này khi bạn đang viết code, bạn sẽ gặp những lỗi đó và bạn sẽ có thể sửa được chúng bất cứ khi nào chúng xuất hiện.

Vì vậy đừng lo lắng về điều này quá nhiều, hãy thay đổi lại điều này và tiếp tục áp dụng tương tự cho các phương thức khác ngoài phương thức khởi tạo.

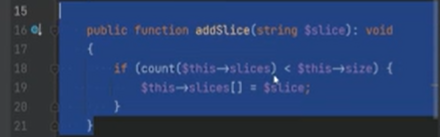
Giả sử rằng chúng ta muốn ghi đè phương thức addSlice, chấp nhận **slice** như một đối số. Nếu chúng ta không gọi phương thức addSlice gốc một cách rõ ràng thì nó sẽ không tự động thực hiện điều đó cho chúng ta. Vì vậy nếu tôi chạy lại cái này, chúng ta sẽ thấy rằng không có gì được in vì nó không tự động gọi phương thức gốc.

Chúng ta cần phải nói rõ ràng gọi nó bằng cách sử dụng addSlice gốc và chuyển bất kỳ đối số cần thiết nào là slice trong trường hợp này. Nếu chúng ta chạy lại nó thì mọi thứ vẫn hoạt động.

Bạn có thể thêm logic tùy chỉnh vào trước khi gọi phương thức cha mẹ hoặc sau đó, nhưng điều này không bắt buộc.

Không cần gọi phương thức cha nếu bạn muốn ghi đè phương thức và có logic tùy chỉnh trong lớp con của mình.

Bạn có thể làm điều đó và bạn có thể thực hiện logic tùy chỉnh ở đây



trong trường hợp này nó sẽ không gọi phương thức cha chút nào khi ghi đè một phương thức trong php, chữ ký phương thức của lớp con phải tương thích với phương thức đặc trưng gốc.

Nếu không, bạn sẽ gặp lỗi nghiêm trọng hoặc nếu bạn đang sử dụng phiên bản php trước php8 thì bạn sẽ chỉ nhận được cảnh báo.

Chẳng hạn trong addSlice, mong đợi chuỗi được cắt đúng.

Nếu chúng tôi thay đổi chuỗi này thành slice số nguyên int



thì điều này sẽ không hoạt động khi chạy.

Bạn thấy rằng chúng tôi đang gặp phải lỗi nghiêm trọng tương tự đối với các kiểu trả về.

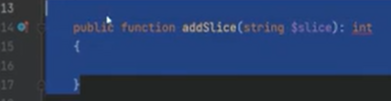
Vì vậy nếu tôi thay đổi chuỗi này trở lại thành string và giả sử chúng ta thay đổi kiểu trả về thành int



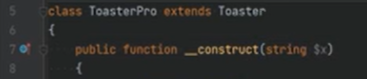
điều này cũng sẽ dẫn đến một lỗi nghiêm trọng vì phương thức đặc trưng không tương thích với phương thức đặc trưng gốc.

Có những thứ như convariance và contravariance tạo ra tham số và các kiểu trả về hơi linh hoạt khi nói đến tính kế thừa. Nhưng đừng lo lắng về điều đó, bây giờ chúng ta sẽ đề cập đến chúng trong một video riêng.

Các quy tắc tương thích không áp dụng cho hàm tạo, nên hãy để tôi xóa điều này khỏi đây



và hãy chấp nhận một số loại tham số trong hàm tạo con để chúng ta có thể làm một cái gì đó giống như chuỗi x



để bạn có thể thấy phương thức đặc trưng của lớp con không giống với phương thức đặc trưng của lớp cha.

Nhưng vì đây là hàm tạo nên các quy tắc đặc trung vẫn ổn, không áp dụng cho các phương thức khởi tạo, bạn cũng có thể chấp nhận các loại đối số khác nhau trong hàm khởi tạo gốc để chúng ta có thể chấp nhận một cái gì đó như int x



và sau đó chúng ta chấp nhận chuỗi y ở đây chẳng hạn và có thể chúng ta chấp nhận int z



sau đó điều duy nhất bạn cần làm là bạn cần đảm bảo rằng bạn đang chuyển các đối số thích hợp cho lệnh gọi hàm tạo gốc.

Nếu bạn đang gọi parent construct, bạn không gọi cái này 

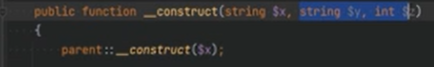
thì bạn không cần phải lo lắng về điều đó.

Nhưng nếu bạn đang gọi parent construct thì bây giờ bạn cần chuyển đối số thích hợp, điều đó tùy thuộc vào cách bạn muốn chuyển này.

Bạn vẫn có thể chấp nhận chuỗi x làm đối số đầu tiên ở đây



và chuyển nó xuống và chỉ chấp nhận 2 đối số bổ sung trong lớp con



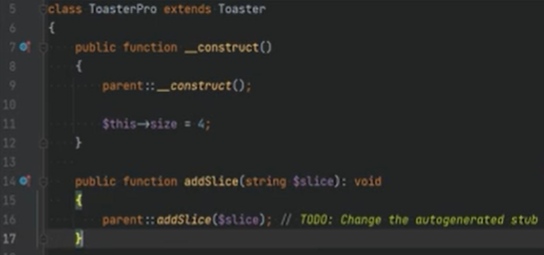
hoặc bạn có thể chỉ cần mã cứng chuỗi này thành một cái gì đó



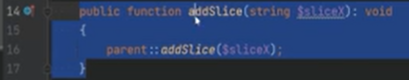
và không chấp nhận nó làm đối số nhưng nó về cơ bản tùy thuộc vào cách bạn muốn xử lý tùy thuộc vào logic ứng dụng và yêu cầu của bạn, hãy xóa phần này khỏi đây và tiếp tục.

Bạn cũng có thể có các tên khác nhau cho các tham số phương thức, điều này sẽ không gây ra vấn đề tương thích nhưng chúng có thể gây ra lỗi nếu tính năng đối số được đặt tên sẽ được sử dụng lại.

Nếu chúng ta ghi đè phương thức addSlice ở đây



và chúng ta gọi nó là slice x và truyền nó vào đây là hoàn toàn hợp lệ,



nó vẫn hoạt động nếu tôi chạy index.php.

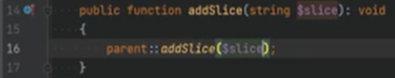
Chúng tôi thấy rằng mọi thứ đều hoạt động nhưng nếu chúng ta đang sử dụng các đối số được đặt tên ở đây,



chúng ta sẽ thấy rằng điều này sẽ không hoạt động nếu chạy lại nó.

Chúng ta gặp lỗi đó là một slice tham số có tên không xác định, vì vậy vì tính năng đối số được đặt tên đã được giới thiệu nên tốt hơn là nên khớp với tên tham số.

Vì vậy chúng tôi sẽ thay đổi điều này thành slice



và mọi thứ sẽ hoạt động, nếu bạn muốn đảm bảo rằng một lớp hoặc một phương thức cụ thể có thể được mở rộng trong trường hợp đó.

Bạn có thể sử dụng từ khóa **final**



nếu bạn xác định một lớp là cuối cùng để bạn có kết quả cuối cùng từ khóa trước từ khóa lớp, điều này khiến bạn không thể mở rộng từ lớp Toaster, vì bạn có thể thấy id của tôi được gạch chân ở đây



và cho tôi biết rằng tôi không thể kế thừa hoặc mở rộng từ lớp cuối cùng nên nếu tôi chạy mã này, chúng tôi sẽ nhận được lỗi nghiêm trọng tương tự như các lớp là cuối cùng.

Bạn cũng có thể có các phương thức là final, vì vậy bạn có thể nói rằng phương thức addSlice là cuối cùng, điều này khiến cho phương thức addSlice không thể bị ghi đè nên nếu tôi chạy nó bây giờ thì chúng ta sẽ gặp một lỗi nghiêm trọng lỗi, vì chúng tôi đang cố gắng ghi đè phương thức cuối cùng. Nếu tôi xóa cái này và chạy nó bây giờ mọi thứ đều hoạt động.

Nên về cơ bản, từ khóa final có thể được sử dụng để ngăn chặn kế thừa lớp và ghi đè phương thức trước khi chúng tôi chuyển sang các ví dụ khác.

Tôi muốn đề cập rằng php có không hỗ trợ đa kế thừa, điều đó có nghĩa là bạn không thể mở rộng nhiều hơn một lớp tuy nhiên nó hỗ trợ kế thừa đa cấp có thể được sử dụng để đạt được kế thừa phân cấp.

Chẳng hạn, nếu chúng ta có một ToasterPro khác có tất cả các tính năng mà ToasterPro có nhưng nó cũng có có một số tính năng bổ sung mà chúng ta có thể mở rộng ToasterPro trong lớp đó và lớp đó sẽ kế thừa tất cả các thuộc tính và hằng số của phương thức public và protected của cả hai lớp ToasterPro và Toaster.

Hãy xem một số ví dụ khác:

Giả sử rằng chúng ta có một lớp giao dịch **transaction** trong đó chúng ta xử lý các giao dịch, ... và nó có tất cả các thuộc tính và phương thức cần thiết để xử lý giao dịch đó.

Giả sử rằng chúng ta cũng có một loại giao dịch khác là giao dịch hoàn tiền **refund transaction** và có thể được biểu diễn dưới dạng một lớp khác.

Vâng, bạn có thể có phương thức hoàn tiền trong lớp giao dịch nhưng điều gì sẽ xảy ra nếu bạn có logic cụ thể về cách bạn hoàn tiền cho các giao dịch của mình trong trường hợp đó.

Giao dịch hoàn trả có thể là một lớp khác có chức năng tùy chỉnh riêng nhưng nó cũng kế thừa tất cả các thuộc tính và phương thức cần thiết từ lớp giao dịch. 

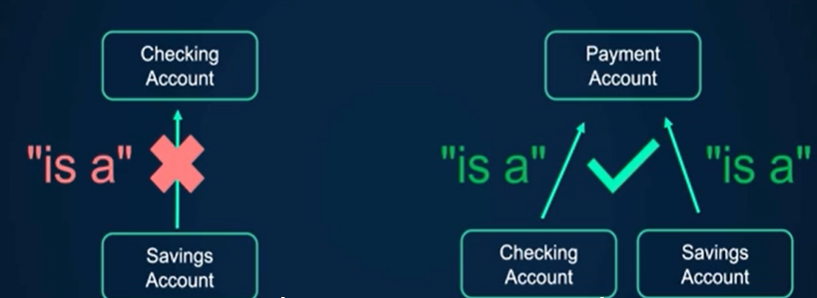
Một ví dụ khác là với tài khoản thanh toán **payment account**.

Bạn có thể có 2 loại tài khoản: tài khoản séc **checking account** và tài khoản tiết kiệm **savings account**



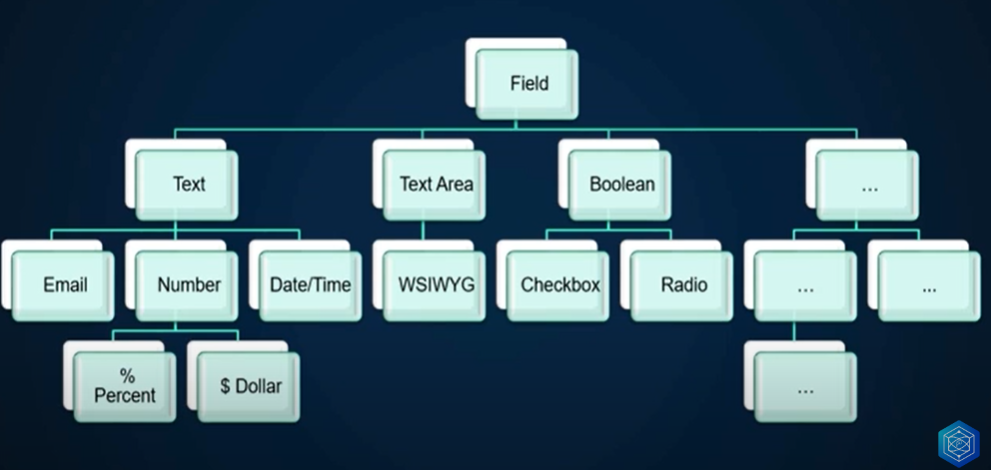
vì tài khoản séc và tài khoản tiết kiệm chia sẻ một số chức năng phổ biến mà bạn có thể có tài khoản thanh toán làm lớp cơ sở và mở rộng tài khoản séc và tiết kiệm từ đó.

Hoặc bạn chỉ cần có một tài khoản séc và sau đó mở rộng tài khoản đó từ tài khoản tiết kiệm nhưng điều đó không thực sự có ý nghĩa vì tài khoản tiết kiệm là không kiểm tra tài khoản và ghi nhớ đó là mối quan hệ và chúng ta sẽ đề cập đến điều đó.



Chút nữa và tôi sẽ giải thích điều đó có nghĩa là gì.

Ví dụ tiếp theo là với các trường biểu mẫu. Vì vậy hãy coi nó như cấu trúc cây



chúng ta có field cơ sở có thể có một số thuộc tính và một số phương thức áp dụng cho field cơ bản và sau đó chúng tôi mở rộng trường cơ sở đó thành trường văn bản thành trường vùng văn bản thành trường loại boolean, v.v. sau đó mỗi trường loại đó có thể mở rộng hơn nữa và chúng tôi có thể có văn bản email chúng ta có thể có trường văn bản số hoặc chúng ta có thể có trường văn bản ngày và giờ. Vẫn có các trường văn bản nhưng chúng có một số chức năng bổ sung, sau đó chúng ta cũng có thể mở rộng trường số hơn nữa sang các trường phần trăm và đô la.

Cũng chỉ là trường các trường số nhưng chúng có một số chức năng bổ sung và điều tương tự cũng áp dụng cho các loại trường khác để bạn có thể thấy đây là một ví dụ về kế thừa đa cấp thay vì kế thừa từ một lớp mà bạn đang kế thừa từ nhiều lớp thông qua hệ thống phân cấp. Vì vậy nó không phải là đa kế thừa mà là kế thừa thông qua hệ thống phân cấp, có nghĩa là trường đô la ở đây kế thừa mọi thứ từ trường số number và sau đó nó cũng kế thừa mọi thứ từ trường văn bản và nó cũng kế thừa mọi thứ từ trường cơ sở nên điều này tổng hợp với các ví dụ.

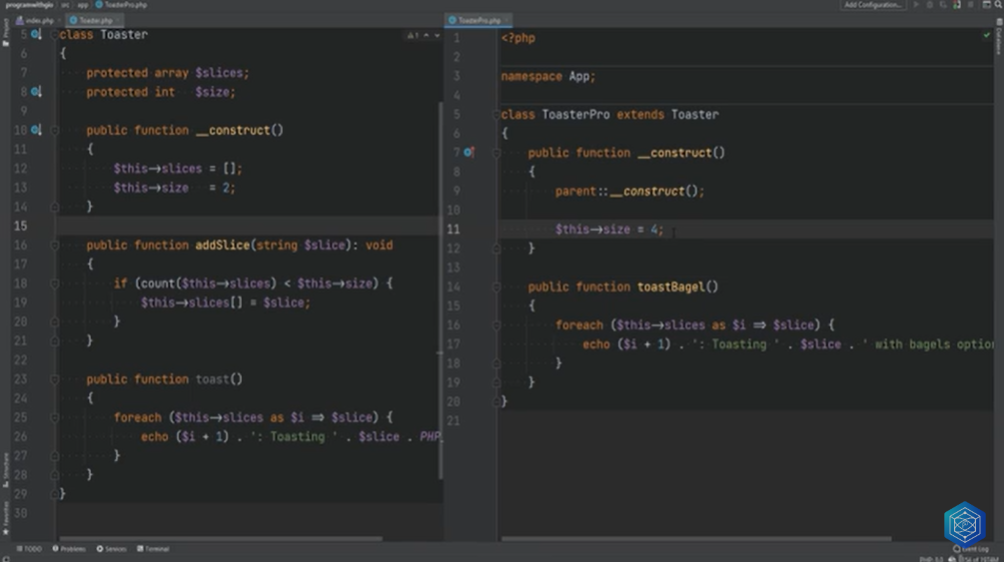
Chúng ta sẽ đề cập đến nhiều trường hợp sử dụng thực tế hơn và nhiều ví dụ thực tế hơn trong khóa học khi chúng ta thực sự làm việc với dự án.

Nhưng hãy nói về lý do tại sao tính kế thừa có thể không phải là một ý tưởng hay.

Tính kế thừa nói chung là tốt vì nó là một phần của nguyên tắc hướng đối tượng nhưng nó cũng có thể xấu khi bị lạm dụng hoặc nếu nó được sử dụng quá mức.

Chẳng hạn như tính kế thừa có thể phá vỡ sự đóng gói vì nó có quyền truy cập vào tất cả các phương thức và hằng số thuộc tính public và protected của lớp cha.

Nên ví dụ như trong trường hợp này tôi có thể trực tiếp thay đổi các thuộc tính của lớp cha



và điều này có thể ổn vì đây là do thiết kế nhưng nó cũng có thể phá vỡ sự đóng gói mà bạn được phép làm điều này bạn có thể làm theo bất kỳ phương thức nào khác.

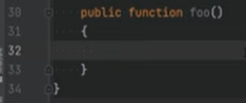
Bạn có thể thay đổi kích thước thuộc tính thành 999



Do đó, một nhược điểm khác của kế thừa là khi bạn kế thừa từ lớp cha, bạn thực sự đang kế thừa tất cả các phương thức public và protected mà bạn thậm chí có thể không cần trong đó.

Ví dụ: Giả sử rằng lớp Toaster có một loại phương thức bổ sung nào đó không áp dụng cho lớp ToasterPro, có thể sau này nó không được dùng nữa hoặc có thể ToasterPro không còn cần phương thức đó nữa.

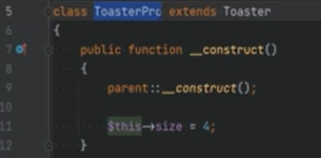
Nhưng lớp Toaster cơ bản có phương thức đó, hãy gọi phương thức này hiện đã đầy đủ **foo**



và giả sử chúng ta đang làm một cái gì đó cụ thể cho Toaster cơ bản khi chúng ta mở rộng Toaster sang ToasterPro, lớp này có quyền truy cập vào phương thức foo

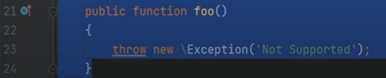


và khi chúng ta khởi tạo ToasterPro ở đây,



chúng ta có thể gọi phương pháp foo đó và điều đó có thể không lý tưởng vì phương pháp này hoàn toàn có thể không áp dụng cho ToasterPro và bạn có thể không muốn cho phép gọi hàm đó trên ToasterPro.

Giải pháp điển hình cho giải pháp mà bạn có thể thấy trong một số cơ sở mã là ghi đè phương thức đó và chỉ cần ném một ngoại lệ.



Theo tôi đây không phải là một thiết kế tốt vì vậy nếu bạn thấy mình đang kế thừa theo cách mà bạn buộc phải ghi đè một phương thức và ném một ngoại lệ theo cách này.

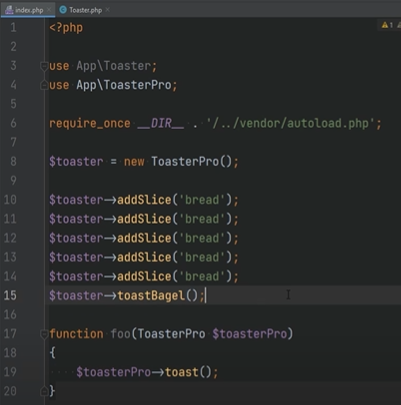
Hãy thử xem xét lại việc sử dụng tính kế thừa trong trường hợp đó.

Bạn có thể muốn tìm đến thứ khác, chẳng hạn như **composision.**

Chúng ta sẽ nói về điều đó chi tiết hơn sau trong khóa học khi chúng ta chuyển sang chủ đề kế thừa so với composision.

Một điều mà tôi đã bỏ lỡ để nói đến là cách bạn thực sự gõ gợi ý và đối tượng của Toaster hoặc ToasterPro khi chấp nhận đối số đó làm đối số cho một hàm hoặc một phương thức khác.

Vì vậy nếu tôi truy cập index.php, tôi sẽ chứng minh điều này bằng cách có một hàm tùy chỉnh ở đây



chứ không phải một phương thức.

Nên tôi sẽ để thực hiện chức năng và giả sử rằng chúng tôi đang chấp nhận ToasterPro và gọi đó là ToasterPro.

Bây giờ chúng ta có thể gọi phương thức đó ở đây

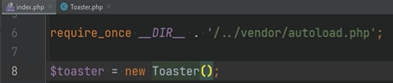


và chuyển toaster làm đối số và hãy loại bỏ



và nó sẽ hoạt động ngay.

Bây giờ nếu chúng ta chuyển vào Toaster thay vì ToasterPro



thì điều này sẽ dẫn đến lỗi vì Toaster không phải là ToasterPro, không mở rộng ToasterPro thay vào đó ToasterPro mở rộng lớp Toaster.

Vì vậy Toaster là máy nướng bánh mì nhưng bản thân máy nướng bánh mì không phải là máy nướng bánh mì chuyên nghiệp.

Cách bạn có thể khắc phục điều này là bạn có thể chỉ cần gõ gợi ý lớp cơ sở là Toaster và phương pháp toaster có sẵn trên cả Toaster vàToasterPro điều này sẽ hoạt động và nó cũng sẽ hoạt động nếu chúng ta chuyển vào ToasterPro.



Vì vậy nếu tôi chạy lại thì bây giờ nó sẽ nướng 4 lát.

Tuy nhiên nếu bạn thử thì sẽ có một vấn đề để gọi bánh mì nướng bagel, cách này sẽ hoạt động nếu bạn chuyển vào đối tượng ToasterPro nhưng nó sẽ không hoạt động nếu bạn chuyển vào Toaster thông thường vì Toaster không tồn tại trong những trường hợp mà bạn có thể thấy trong một số cơ sở mã nơi họ kiểm tra xem phương thức toastBagel tồn tại.

Nếu đối tượng là phiên bản của toasterPro thì hãy gọi phương thức đó, nếu không thì gọi phương thức toast cơ bản thông thường nhưng theo tôi đó là thiết kế tồi và có nhiều cách tốt hơn để thực hiện điều đó bằng cách sử dụng thứ gọi là composition và chúng ta sẽ nhận được với điều đó ở phần sau của khóa học này.

Tôi đã đề cập đến mối quan hệ trước đây rằng ToasterPro về cơ bản là một máy nướng bánh mì và bạn có thể sử dụng mối quan hệ đó để quyết định xem bạn có thực sự cần quyền kế thừa hay không.

Nếu bạn có một lớp mà bạn có thể áp dụng là mối quan hệ sau đó bạn có thể tận dụng sức mạnh của sự kế thừa trong trường hợp của chúng tôi, ToasterPro là một Toaster, đó là lý do tại sao tính kế thừa có thể hoạt động nếu mối quan hệ không có ý nghĩa hoặc nó không áp dụng với lớp sử dụng tính kế thừa của bạn, thì bạn có thể đang sử dụng tính kế thừa trong đó

Nên sử dụng composition thay vào đó, lời khuyên của tôi là đừng đạt đến quyền thừa kế trong mọi trường hợp, hãy đạt đến nó khi bạn có mối quan hệ dễ dàng thích hợp giữa con và cha mẹ.

Vì vậy khi nào bạn sẽ không đạt được quyền thừa kế, hãy nói rằng sau này bạn phải triển khai một trình mô phỏng cho một trong những lò nướng ưa thích còn có chức năng nướng bánh mì và bánh mì tròn.

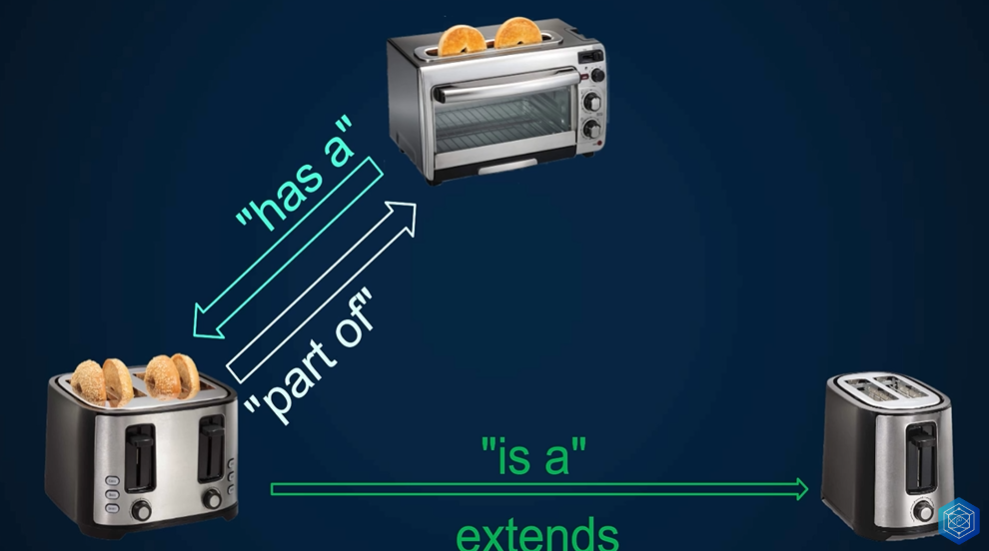


Bạn có thể muốn triển khai lò nướng đó bằng cách mở rộng từ lớp máy nướng bánh mì hoặc đơn giản là mở rộng từ lớp chuyên nghiệp về máy nướng bánh mì để kế thừa tất cả chức năng chuyên nghiệp của máy nướng bánh mì và máy nướng bánh mì, sau đó thêm chức năng bổ sung mà lò nướng đó hỗ trợ nhưng chúng sẽ gây ra nhiều vấn đề hơn trong khi duy trì và bổ sung thêm nhiều chức năng hơn cho ứng dụng của bạn.



Vì vậy nếu bạn nghĩ theo cách khác nếu lò nướng không phải là máy nướng bánh mì và không đủ điều kiện thì đó là một mối quan hệ .

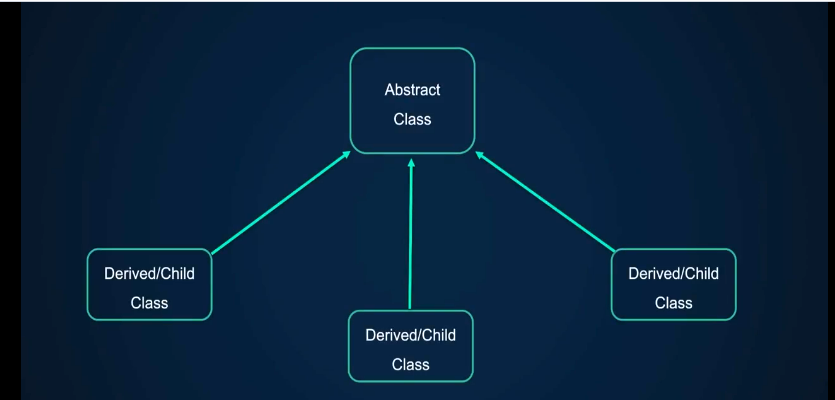
Làm thế nào chúng ta có thể kết hợp máy nướng bánh mì vào lớp lò nướng của mình? Mối quan hệ với tôi có vẻ như lò nướng có máy nướng bánh mì hoặc máy nướng bánh mì là một phần của lớp lò nướng phải không

chúng ta chỉ cần sao chép máy nướng bánh mì chuyên nghiệp ở đây và đặt tên cho nó là lò nướng ưa thích

Và bây giờ chúng ta hãy đóng nó lại và chúng ta 'không mở rộng từ bất cứ thứ gì và hãy xóa tất cả các phương pháp này và sau đó có thể chúng ta đã có một loại phương pháp nào đó ở đây có thể chiên mọi thứ và sau đó chúng ta có một phương pháp ở đây để nướng mọi thứ và sau đó chúng ta có một phương pháp nướng bánh mì tròn ngay bây giờ như bạn nhớ bánh mì nướng và bánh mì nướng bagel đã tồn tại trên lớp máy nướng bánh mì chuyên nghiệp, chúng tôi có nó ngay tại đây và bánh mì nướng tồn tại trên lớp máy nướng bánh mì thay vì mở rộng máy nướng bánh mì chuyên nghiệp ở đây, chúng ta có thể chỉ cần sử dụng thứ gọi là thành phần trong đó lò nướng ưa thích có chức năng nướng bánh mì chuyên nghiệp và chúng ta có thể làm rằng bằng cách đơn giản sử dụng các thuộc tính để chúng ta có thể làm máy nướng bánh mì chuyên nghiệp riêng tư và chúng ta có thể chấp nhận điều đó trong hàm tạo như một phần phụ thuộc và để làm cho điều này tốt hơn nữa, chúng ta có thể sử dụng quảng cáo thuộc tính ở đây để chúng ta có thể loại bỏ hoàn toàn điều này và bây giờ chúng ta có máy nướng bánh mì vì vậy thay vì sao chép phương thức nướng bánh mì ở đây, nơi chúng ta sẽ sao chép chức năng bánh mì nướng và bánh mì nướng bagel, chúng ta có thể chỉ cần gọi cái này là bánh mì nướng và bánh mì nướng bánh mì nướng này để bạn có thể thấy chúng tôi không sử dụng tính kế thừa nhưng chúng tôi vẫn có thể sử dụng ưu điểm của máy nướng bánh mì chức năng bằng cách sử dụng thứ gọi là composition, chúng ta sẽ nói nhiều hơn về bố cục ở phần sau của khóa học.

**2.10. Abstract Classes & Methods: lớp trừu tượng và phương thức trừu tượng**

Một lớp trừu tượng giống như một khuôn mẫu (template) hoặc lớp cơ sở mà lớp con (child class) hoặc các lớp cụ thể (concrete class) có thể mở rộng.



Điều này được thực hiện bằng cách tận dụng sức mạnh của thừa kế, như đã nêu trong bài học trước (2.9).

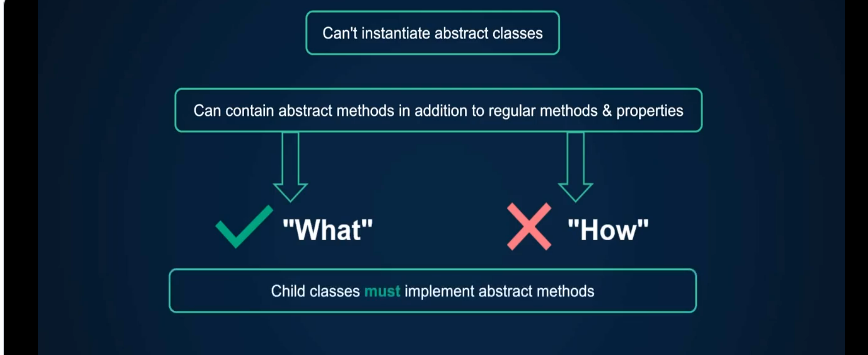
Có một số quy tắc khi làm việc với lớp trừu tượng và các phương pháp trừu tượng.

Một lớp trừu tượng không thể được khởi tạo. VD: bạn chỉ có thể mở rộng nó, có nghĩa là bạn không thể tạo các đối tượng trực tiếp từ lớp trừu tượng.

Một lớp trừu tượng có thể có các phương thức trừu tượng chỉ chứa phần khai báo ( method signature) hoặc định nghĩa (definiton) mà không có thực hiện cụ thể.

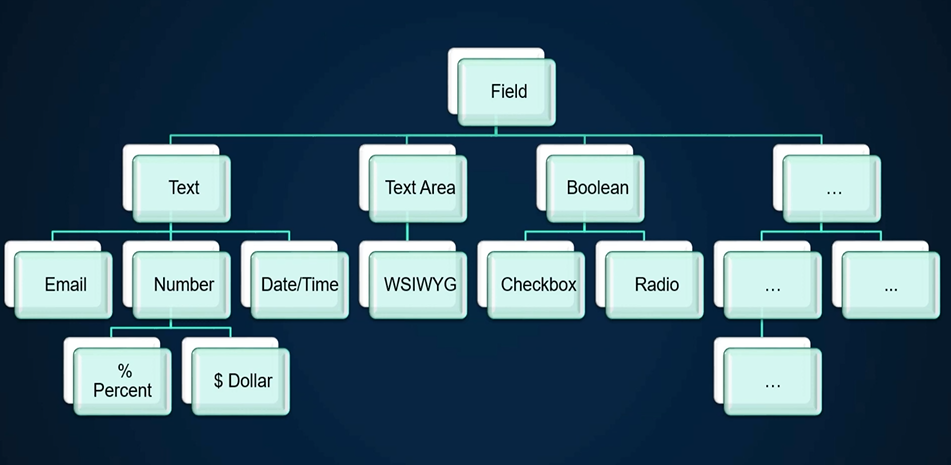
Nói 1 cách đơn giản, lớp trừu tượng biết “cái gì” (what), nhưng nó không biết “làm thế nào” (how).

Phần “how” được triển khai trong các lớp con.



Trách nhiệm của lớp con là triển khai (implement) các phương pháp trừu tượng này.

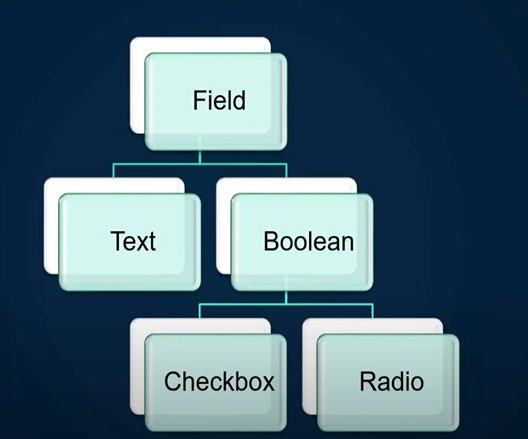
Hãy xem ví dụ này từ bài học trước.



Chúng ta có trường cơ sở (base filed) ở đầu

và sau đó chúng ta có trường văn bản (text filed), trường vùng văn bản (text area) và trường boolean mở rộng từ trường cơ sở, và sau đó chúng ta tiếp tục phân cấp xuống.

Hãy triển khai phiên bản đơn giản hơn của nó,



trong đó chúng ta chỉ có trường cơ sở,

sau đó chúng ta sẽ có trường văn bản và boolean mở rộng từ trường cơ sở,

và sau đó chúng ta sẽ có trường checkbox và các trường nút radio (radio button) mở rộng từ trường Boolean.

Tôi đã tạo các lớp này được tạo ở đây,



có lớp trường cơ sở, tôi có lớp văn bản, lớp boolean, hộp kiểm (checkbox) và tôi có nút radio.

Hiện tại chúng chỉ là các lớp rỗng, vì vậy hãy điền chúng vào.

Trước khi chúng ta tieps tục với các lớp trừu tượng, chúng ta hãy chỉ sử dụng kế thừa thông thường.

Trường cơ sở có thể có một số thuộc tính và các phương thức được chia sẻ bởi tất cả các trường.

Tất cả các trường có thể có thuộc tính “name”.

Ít nhất là tất cả các trường mà chúng ta có ngay tại đây, trường văn bản và hộp kiểm và các nút radio, tất cả đều có các thuộc tính “name”.

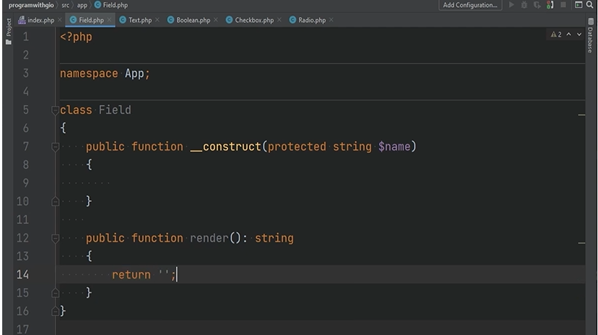
Chúng ta có thể định nghĩa “name” là một trong những thuộc tính.

Tôi sẽ tạo 1 **construct()** và tôi sẽ sử dụng tính năng **property promotion** ở đây.

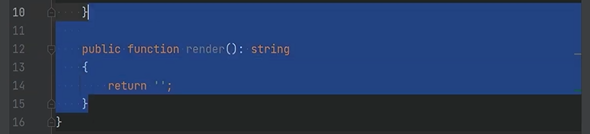
Chúng ta sẽ có **contruct(protected string $name)** và sau đó chúng ta sẽ có một phương thức để render trường.

Chúng ta có thể có 1 hàm public trả về kiểu string và trường cơ sở này chỉ đơn giản trả về

một chuỗi trống vì chúng ta chưa biết phải render cái gì.



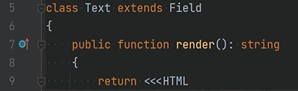
Hãy sao chép cái này và đưa nó vào các trường khác.



Ngoài ra, chúng ta có thể mở rộng (extend) trường cơ sở ở đây và hãy đặt nó vào để ghi đè (override) lên phương thức kết xuất (render).



Tôi sẽ sử dụng cú pháp Dox ở đây để trả về HTML.



Chúng ta sẽ có thẻ input, type, text, và sau đó là name, và chúng ta có thể đặt “name” vào đó và sau đó đóng thẻ.



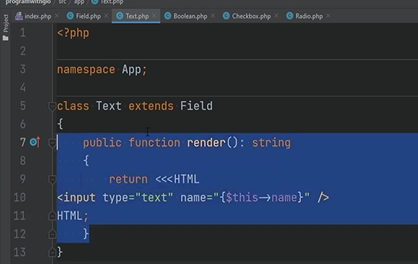
Bây giờ, tất nhiên, bạn không phải sử dụng cú pháp Dox trong ứng dụng thực.(<<<HTML)



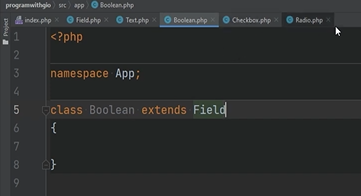
Bạn có thể sẽ sử dụng một tập tin view và render nội dung thành một chuỗi và trả về chuỗi đó thay vì trả về HTML như vậy.

Nhưng hãy giữ mọi thứ đơn giản cho ví dụ này.

Tôi sẽ sao chép cái này và đưa nó vào các trường khác.



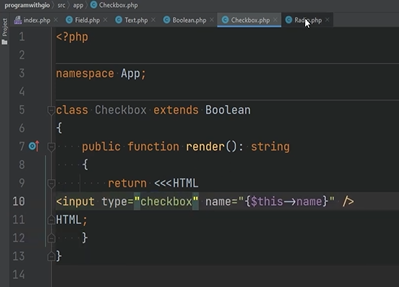
Trong lớp boolean, chúng ta có thể chỉ cần extend trường cơ sở.



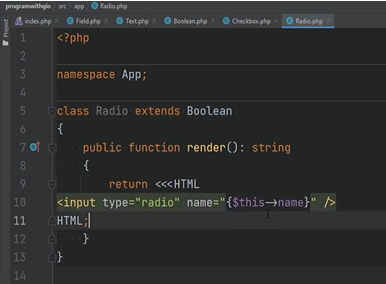
Bởi vì chúng ta không biết làm thế nào để render một trường Boolean. Thay vào đó, chúng ta sẽ ghi đè lên phương thức render trong các trường con của boolean là checkbox và nút radio.

Chúng ta đi đến Checkbox và mở rộng từ lớp boolean, sau đó chúng ta ghi đè lên phương thức render

và chúng ta chỉ cần đặt type là checkbox.



chúng ta đi đến radio mở rộng boolean, và chúng tôi thay đổi type thành radio.



Bây giờ, tất nhiên, có nhiều cách khác nhau để thực hiện những điều như vậy.

Ví dụ: thay vì có nhiều lớp, bạn chỉ cần lưu trữ type vào một thuộc tính và render nó theo cách đó.

Nhưng bởi vì mỗi trường riêng lẻ có thể có logic tùy chỉnh riêng nó,

tốt hơn để tách các mối quan tâm và đặt chúng trong các lớp riêng của nó, ngay cả khi bạn đang lặp lại một ít code.

Bằng cách đó bạn có thể linh hoạt hơn và nhiều không gian hơn để thêm vào các tính năng cụ thể của trường cho lớp cụ thể đó.

Ngay bây giờ, chúng ta chưa sử dụng abstract.

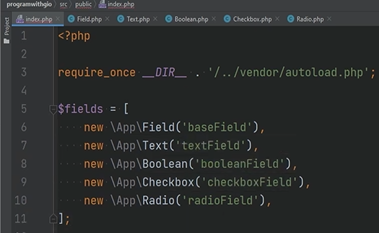
Hãy đi đến index.php và tạo một mảng các trường, mảng đó sẽ chứa một phiên bản của từng trường đó.

Hãy tạo một đối tượng của trường cơ sở và truyền vào một tên bất kỳ.

Chúng ta sẽ gọi trường cơ sở này.

Hãy sao chép trường này 4 lần và hãy thay đổi chúng thành các trường tương ứng.

Bây giờ chúng ta có 1 mảng gồm 5 trường.



Bây giờ chúng ta hãy xem qua từng trường này và chúng ta sẽ render và in ra HTML.

Hãy thêm một đường ngắt ở đây và hãy làm mới trang và chúng ta thấy một hộp nhập,

Chúng ta thấy một checkbox và một nút radio.

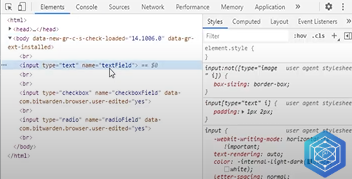
Nó dường như đang hoạt động.



Chúng ta thấy có khoảng trống cho trường cơ sở và chúng ta có khoảng trống cho trường Boolean.

Nếu chúng ta mở DevTools, chúng ta thấy rằng tên đã được đặt đúng.

Nó được đặt thành trường text, trường checkbox và trường radio.



Nhưng có một vấn đề ở đây.

Chúng ta có thể tạo các phiên bản của trường cơ sở và trường boolean cơ sở trực tiếp.

Đó là điều mà chúng ta không muốn cho phép vì chúng ta không biết làm thế nào

để render trường cơ sở và chúng tôi không biết cách render trường Boolean.

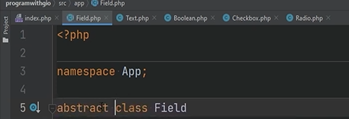
Chúng ta chỉ biết làm thế nào để render các lớp cụ thể, biết cách render text, biết làm thế nào để render checkbox và radio.

Nhưng hai trường này là các trường cơ sở hoặc template cung cấp một số chức năng cho tất cả các trường này mở rộng từ chúng.

Đó là nơi bạn có thể sử dụng các lớp trừu tượng.

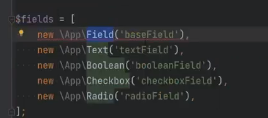
Nếu chúng ta đi đến lớp flied, chúng ta có thể thay đổi lớp này thành một lớp trừu tượng.

Chúng ta có thể làm điều đó bằng cách sử dụng từ khóa “abstract” ngay trước class.

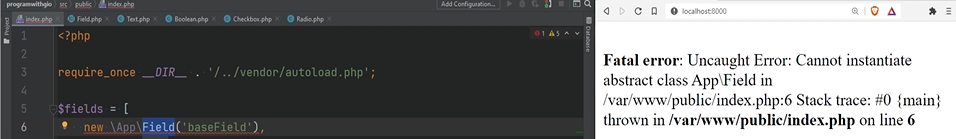


Bây giờ chúng ta không còn có thể tạo các đối tượng của lớp Flied nữa.

Nếu chúng ta đi đến index.php, chúng ta thấy rằng nó đã được gạch chân ngay tại đây.



Nếu tôi làm mới, chúng ta sẽ nhận được lỗi nghiêm trọng.



Hãy xóa nó khỏi đây và hãy tiếp tục.

Một điều khác chúng ta có thể làm ở đây là phương thức render đơn giản chỉ trả về một chuỗi rỗng.

Bây giờ, như tôi đã đề cập ở phần đầu bài học: trong bài học, bạn có thể có các phương pháp trừu tượng. Một phương pháp trừu tượng chỉ có định nghĩa mà không cài đặt.

Đây là một ví dụ hoàn hảo cho điều đó.

Chúng ta biết rằng tất cả các trường này cần phải có phương thức render, nhưng chúng ta không biết chúng thực sự được triển khai như thế nào.

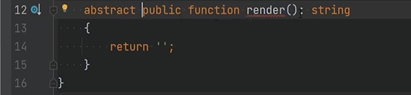
Điều đó nên tùy thuộc vào lớp con để xác định cách nó thực sự được thực hiện.

Đó là nơi chúng ta có thể sử dụng các phương thức trừu tượng bởi vì ngay bây giờ, chúng ta có thể đơn giản để trống trường này và trường văn bản sẽ không được hiển thị và điều đó không lý tưởng.

Chúng ta muốn buộc các lớp con phải cài đặt phương thức render.

Để làm điều đó, chúng ta có thể sử dụng các phương thức trừu tượng.

Chúng ta có thể đến đây và viết trước public, chúng ta có thể sử dụng **abstract public function**.



Nó được gạch chân ở đây bởi vì bạn chỉ có thể có phương thức đặc trưng, bạn không thể có cài đặt.

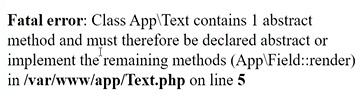
Chúng ta cần loại bỏ việc triển khai và đặt dấu chấm phẩy ngay tại đây.



Bây giờ, điều này có nghĩa là tất cả các lớp mở rộng từ lớp filed cần triển khai phương thức render.

Nếu bạn đi đến text class bây giờ, nó gây ra lỗi cho chúng ta vì chúng ta đã xóa phương thức kết xuất.

Nếu tôi làm mới trang, chúng tôi sẽ gặp lỗi nghiêm trọng.

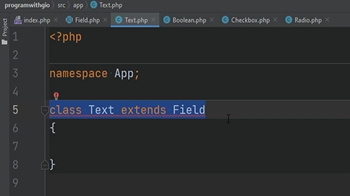


Lỗi đó là lớp văn bản chứa một phương thức trừu tượng và do đó nó cần được khai báo là lớp abstract hoặc cài đặt các phương pháp còn lại.

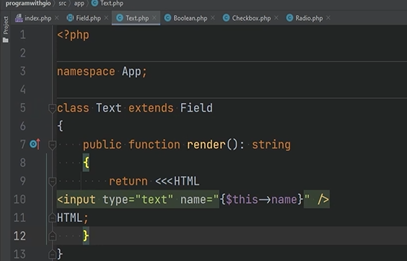
Điều đó có nghĩa là nếu bạn không cài đặt phương pháp trừu tượng, thì bạn cần đánh dấu lớp là trừu tượng.

Nếu bạn có ít nhất một phương pháp trừu tượng, bản thân lớp đó cần được khai báo là abstract.

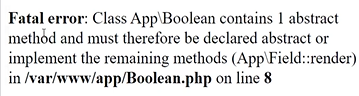
Bạn có một phương pháp trừu tượng ở đây bởi vì bạn kế thừa tất cả các phương thức public và các phương thức protected từ lớp trường cơ sở.



Do đó, vì lớp trường có phương pháp trừu tượng, chúng ta cần khai báo class này là abstract hoặc chúng ta cần thực hiện phương thức render.



Trong trường hợp này, chúng ta sẽ chỉ cần đưa lại phương thức render và chúng ta làm mới trang và chúng ta vẫn gặp lỗi này.



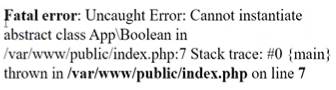
Nhưng bây giờ lỗi đến từ lớp Boolean. Nếu chúng ta chuyển sang Boolean, chúng tôi thấy rằng chúng ta chưa bao giờ cài đặt phương thức kết xuất ở đây.

Trong trường hợp này, chúng ta không cần phải thực hiện phương thức render bởi vì, một lần nữa, chúng ta không biết cách hiển thị các trường Boolean.

Tùy thuộc vào các lớp cụ thể để tìm ra cách hiển thị các trường của riêng chúng.

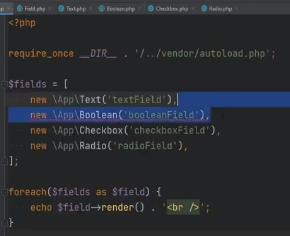
Checkbox phải tìm ra cách render checkbox , v.v…

Trong trường hợp này, chúng ta cũng có thể đánh dấu lớp này là trừu tượng.Nếu tôi làm mới trang, chúng tôi sẽ gặp lỗi nghiêm trọng.



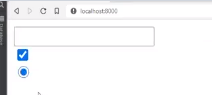
Đó là bởi vì chúng ta đang cố gắng tạo một đối tượng của lớp boolean trong index.php

Chúng ta cũng cần xóa cái này.



Mọi thứ đã trở lại bình thường.

Bây giờ chúng ta có trường văn bản, trường hộp kiểm và nút radio.



Bởi vì chúng ta đang sử dụng kế thừa ở đây, tất cả các lớp con phải tuân theo các quy tắc kế thừa thông thường và tương thích đặc trưng.

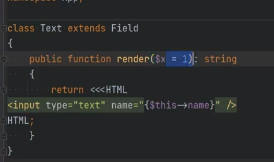
Chúng ta đã thảo luận điều này trong bài học trước, nhưng tôi quên đề cập đến một điều là các phương pháp mà bạn đang ghi đè trong lớp con thực sự có thể có các đối số bổ sung với các giá trị mặc định.

Điều này cũng áp dụng cho việc cài đặt các phương pháp trừu tượng.

Ví dụ: phương thức render, nó không chấp nhận bất kỳ đối số nào.

Nhưng bạn có thể ghi đè lên nó và chấp nhận một số đối số với các giá trị mặc định.

Bạn có thể nói X bằng một, và điều này là hợp lệ.



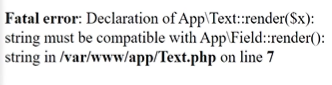
Nếu tôi làm mới trang, mọi thứ vẫn hoạt động.

Tuy nhiên, nếu bạn không định nghĩa các giá trị mặc định



thì điều này không còn tương thích nữa và nó sẽ thất bại vì phương thức cha không có bất kỳ đối số nào và phương pháp con đang yêu cầu một đối số và điều đó không hoạt động.

Nếu tôi làm mới, chúng ta sẽ nhận được lỗi nghiêm trọng.



Một điều khác bạn nên biết là phạm vi của phương pháp trừu tượng đó có thể là public hoặc protected. Bạn không thể đặt tùy chọn này thành private.

Điều đó có ý nghĩa là bạn không thể ghi đè lên các phương thức private.

Khi bạn khai báo các phương thức là abstract, bạn cần cài đặt phương thức đó trong lớp con và chúng ta không thể ghi đè lên phương pháp private trong lớp con và do đó nó sẽ không hoạt động.

Đó là lý do tại sao bạn cần phải có nó là protected hoặc public.

Khi nào bạn sẽ sử dụng một lớp trừu tượng?

Bạn có thể sử dụng một lớp trừu tượng khi bạn muốn buộc các lớp con phải cài đặt các phương thức trong khi bạn cung cấp một số chức năng cơ bản.

Các phương thức không phải là trừu tượng có thể được sử dụng trên tất cả các lớp con.

Sau đó, bạn có thể có một số thuộc tính chung áp dụng cho tất cả các lớp con.

Bây giờ, nếu bạn có một lớp trừu tượng.

Bạn có một loạt các phương thức trừu tượng, thì có lẽ bạn nên xem xét không

sử dụng abstract và thay vào đó tiếp cận với một thứ gọi là **interfaces** (giao diện).

Chúng ta sẽ nói về giao diện trong bài học tiếp theo.