Learn PHP

Going to Bed

This project provides practice for:

* Creating objects from classes
* Writing and using methods
* Using static methods

**Tasks**

16/17 Complete

Mark the tasks as complete by checking them off

**Make a Helper Class**

1.

In this project we will be creating some pajamas for barnyard animals. The farmer has a special way that she likes names formatted. To ensure this, we are going to build our own string function similar to PHP’s built in strtolower or strtoupper.

We’re going to put this function inside of a helper class StringUtilities (making it a method). We can then add other string utilities or reuse this class in another project for the same farmer.

Begin by creating a StringUtilities class.

class StringUtils  
{  
   
}

2.

The farmer likes names to be entirely lowercase, except for the second letter, which should always be uppercase.

Define a method called secondCase inside of StringUtilities which accomplishes this. This method should be static since we won’t be instantiating StringUtilities. We are only using it as a helper class.

We defined a public method that accomplishes the intended task:

public static function secondCase($string) {  
  $string = strtolower($string);  
  $string[1] = strtoupper($string[1]);  
  return $string;  
}

3.

Test this method by invoking it statically with some strings and printing the results to the screen. There is a case that causes an error, can you find it?

Try:

php echo StringUtils::secondCase("MCDONALD"); echo StringUtils::secondCase("baldwin"); echo StringUtils::secondCase("Q");php

4.

While “Q” might not be a popular name, we might use StringUtils in a different application in the future and our method should handle single characters gracefully.

Modify secondCase so that it does not generate an error for:

echo StringUtils::secondCase("Q"); #should print "q"  
echo StringUtils::secondCase(""); #should print an empty string

There are a couple of ways to accomplish this. We wrapped the upper casing in a conditional:

if (strlen($string) >= 2) {  
  $string[1] = strtoupper($string[1]);  
}

**Pajamas Already**

5.

Now that we have our helper class, we are ready to start making pajamas.

Create a Pajamas class. It should have 3 private properties:

* owner
* fit
* color

You can declare the properties on 3 separate lines or combine them into one statement like this:

class Pajamas  
{  
  private $owner, $fit, $color;  
}

6.

Add a constructor class for Pajamas. It should allow 3 arguments to be passed in corresponding to the 3 properties we just defined.

Provide defaults for each argument (of your choosing).

Make sure to assign the passed in values to their corresponding properties. Use secondCase in the StringUtilities helper class to ensure that the owner property is properly formatted.

The argument names don’t have to match the property names, but it can be a bit more straightforward:

function \_\_construct(  
  $owner = "unclaimed",  
  $fit = "fine",  
  $color = "white"  
) {  
  $this->owner = StringUtils::secondCase($owner);  
  $this->fit = $fit;  
  $this->color = $color;  
}

7.

We want to be able to describe each Pajamas object.

Add a public describe method which returns a string using the 3 properties to tell us about the object.

We went with:

public function describe() {  
  return "$this->owner's $this->color pajamas fit $this->fit.";  
}

8.

Now we have enough that we can begin testing our class out. Create an instance of the Pajamas class for “CHICKEN”. Assign it to the variable chicken\_PJs. Use your favorite color and fit.

Print the result of using the describe method on chicken\_PJs.

Depending on your implementation of describe, this code:

$chicken\_PJs = new Pajamas("CHICKEN", "just right", "purple");  
echo $chicken\_PJs->describe();

Should print something like:

cHicken's purple pajamas fit just right.

9.

Let’s say something happened to chicken’s pajamas to change the fit. Like maybe they were washed many times and shrunk. Since fit is a private property, we need to make a setter.

Create a public method setFit which can be used to modify the fit of the pajamas.

We can assign the passed in value to the fit property:

public function setFit($new\_fit) {  
  $this->fit = $new\_fit;  
}

10.

Now test the setFit method by tightening up chicken’s PJs. Be sure to print the result.

We tested it with:

echo "\n...they wash their PJs many times....";  
$chicken\_PJs->setFit("a little tight");  
echo "\n";  
echo $chicken\_PJs->describe();

**A Specialized Type of Pajamas**

11.

Some animals need their pajamas to have buttons so that they can put them on.

Create a new class ButtonablePajamas that inherits from Pajamas.

class ButtonablePajamas extends Pajamas  
{  
   
}

12.

We’ll need a new private property, button\_state, to maintain the status of the buttons (“buttoned” or “unbuttoned”).

Add this property to ButtonablePajamas and set it to "unbuttoned".

private $button\_state = "unbuttoned";

13.

The describe method makes no mention of buttons, but it seems like the right place for a message about the buttons.

Within ButtonablePajamas, override the describe method. The version for this class should include a statement about whether the pajamas are buttoned or not.

Remember that you can reuse the parent method with parent::describe() and concatenate the statement about the buttons.

public function describe() {  
  return parent::describe() . " They also have buttons which are currently $this->button\_state.";  
}

14.

Let’s test this class on an animal that definitely needs buttons. Create an instance of ButtonablePajamas with the owner "moose" and save it to the variable $moose\_PJs.

Use the describe method to ensure that it let’s us know the pajamas are unbuttoned.

We chose “kind of loose” as the fit and “red” as the color.

$moose\_PJs = new ButtonablePajamas("moose", "kind of loose", "red");  
echo "\n";  
echo $moose\_PJs->describe();

15.

Moose would like to close his pajamas, but button\_state is private. Create a method toggleButtons that flips the state of the buttons. It should take no arguments.

There are a few ways to approach this. Here we are checking the button\_state and switching it to the opposite state:

public function toggleButtons() {  
  if ($this->button\_state === "unbuttoned") {  
    $this->button\_state = "buttoned";  
  } else {  
    $this->button\_state = "unbuttoned";  
  }  
}

16.

Make sure to check that this method works as expected. Print the result.

Since the pajamas start unbuttoned, the following code should button the pajamas and let us know:

$moose\_PJs->toggleButtons();  
echo "\n";  
echo $moose\_PJs->describe();

**Beyond**

17.

Congratulations on creating some object oriented pajamas!

If you’d like additional practice, you can keep expanding the project. Here are some ideas:

* A class of pajamas that has a hat
* Performing validation on setFit to only allow “tight”, “fine”, and “loose”
* Setter and getter methods for the owner and color properties