asgn02 -- Inheritance and object access control

Objectives

- Write classes that inherit behaviors (properties and methods) from a parent class.
- Use object access controls: public, protected, and private.
- Fix errors with git restore.
- Create a `dev` branch.
- Merge the `dev` branch with the `main` branch.
- Create a VSCode Workspace.

Watch the LiL videos

Watch the following two chapters

- Chapter 3 - Class Inheritance
- Chapter 4 - Object Access Control

from the LinkedIn/Learning tutorial, PHP: Object-oriented Programming

Set up

- Create a folder named **asgn02-inheritance** inside your **web250** folder.
- Inside your **asgn02-inheritance** folder, create files named
    - challenge-02-inerhitance.php
    - inheritance.php
    - challenge-03-oac.php

We are still embedding our classes in one file. We will separate the files into their own classes soon.

Git

Continue using git for your **web250** folder.

Open GitBash or your terminal and navigate to your **web250** folder.

[Win]

```
cd c:\xampp\htdocs\web250
```

[Mac]

```
cd /Applications/MAMP/htdocs/web250
```

Start with the following commands.

```
git status
git add .
git commit -m"Starting asgn02-inheritance"
```

Create a branch

In the past assignment, we worked on the main branch. The main branch is typically used for code that is ready for production, and, technically, we are still in the development phase. Branches are helpful for bug fixes, features, etc.

Watch the video (4:00) and read the docs on Git branches from Atlassian. Note that his video shows the **master** branch. The **master** branch is an older term. The newer term is **main**

In this example, I am demonstrating the step-by-step branching process. In the video, the author shows the shortcut.

Here is an example of the shortcut

First, see what branches are available.

```
git branch
```

You should only have **main** at this point.

Create a branch called **dev**

```
git branch dev
```

This creates the branch. Now we want to **checkout** to the new branch.

```
git checkout dev
```

You can work safely on the **dev** branch without messing up your code on the **main** branch.

At the end of the assignment, we will merge our new **dev** branch to our **main** branch.

Class Inheritance

Watch all the Class Inheritance videos. It helps to watch the entire chapter before coding the second time.

The Challenge

Complete the challenge at the end. Choose your subject, as Kevin Skoglund suggests. I have found that anything with a solid taxonomy is a good candidate for inheritance. Mr. Skoglund mentions animals, in addition, other similar topics such as birds or plants would make for good subjects. There are excellent bird resources at Cornell Lab of Ornithology and the National Audubon Society. Be careful not to make this too big -- it can grow out of control quickly.

If you choose a bird as your category, you can start with the class we created in the previous lesson.

This isn't easy. Make sure you give it some time.

Your Challenge Code Must

- Have one parent class and at least two subclasses.
- Classes must contain at least one class variable and one class method.
- Demonstrate that the subclass inherits from the parent class.

Git Merge

Once your code is complete, it is time to merge your **dev** branch with the **main** branch. Do it by checking out to the **main** branch, then perform the merge. First you need to stage and commit.

```
git status (I use git status a lot)

git add . (stage the files)

git status (green files are staged)

git commit -m "Completed the inheritance challenge" (commit the files)

git status (same as before)

git checkout main (move to the main branch)

git branch (see what branch you are using - it should be **main**)

git merge dev (merges the code from your **dev** branch to your **main** br

git log (see your git history)
```

Checkout to the dev branch to continue working

```
git checkout dev
```

Object Access Control Challenge

Watch the entire Object Access Control tutorial (chapter 4). Use the **challenge-03-oac.php** file for this code. You must address *all* of Kevin's points.

- Add visibility modifiers to the bicycle class
- Set visibility for all existing properties and methods. Deciding what to make public, protected, and private is difficult. To give yourself some help, check out the answer for visibility, then try to do it yourself.
- Create a **unicycle** subclass
- Add the property **$wheels** and set values for each class.
- Define a **wheel_details()** method which returns "it has two wheels" when called
- Make **weight_kg** a private property.
- Define a **set*weight*kg()** method (setter method).
- Create a **getter** method to read that value back followed by "kg."

- Modify the **weight_lbs()** method to add "lbs" to it.
- What bug have you introduced to the **$weight_kg**?

Try doing as much of this as possible without peeking at the solution. It's an excellent exercise.

Git

Once finished, then it is the same process with git as before.

```
git status (I use git status a lot)

git add . (stage the files)

git status (green files are staged)

git commit -m "Completed the inheritance challenge" (commit the files)

git status (same as before)

git checkout main (move to the main branch)

git branch (see what branch you are using - it should be **main**)

git merge dev (merges the code from your **dev** branch to your **main** br

git log (see your git history)
```

Checkout to the dev branch to continue working git

```
git checkout dev
```

Add Object Access Controls to Your Inheritance Exercise Code

- Modify the code you wrote for the **inheritance** exercise.
- Use the same principles Mr. Skoglund used to improve it by adding visibility modifiers (public, protected, private).
- Try using the private modifier for a class variable in the superclass and see what happens in the subclass.

- Try using the protected modifier for a class variable in the superclass and see if that corrects the problem.
- Add setters and getters to your code. This seems like overkill (some of it is). Setters and getters protect your objects. The best time to use them is when you want to set default values or validate incoming values.

  At first, this will seem like overkill for such a small program (it is), but it is a good starting point. Remember, one of the key concepts is to set class variables as private and the setter and getter functions as public.

Git

- Same process as before.
- Take your time to keep track of your branches!
- You need to stay on the **main** branch to push your code to your GitHub repo

GitHub

Your code is ready for production (technically not ready for production, but we are at an excellent point to pretend); push your code to your GitHub repo.

Submit your work

Copy the URL for your GitHub account and post it in the Comments section of Moodle.