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.
- Create a new branch named `asgn02-inheritance` branch in git.
- Merge the `asgn02-inheritance` branch with the main branch.

Clean code

Kevin's code does not follow the coding standards that we use at A-B Tech. His code follows the PHP PS2 recommendations. For this and all future assignments, we will use **camelCase** instead of underscores for our variable and function names. We will change Kevin's code.

Watch the LinkedIn/L videos

Watch the following two chapters from the LinkedIn/Learning tutorial, PHP: Object-oriented Programming.

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

Set up

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

We are still embedding our classes in one file. We will break them out to their own files 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.

Branches are extremely 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. Either one is acceptable.

First, see what branches are available.

```
git branch
```

You should only have main at this point. Create a branch called `asgn02-inheritance`

```
git branch asgn02-inheritance
```

This creates the branch. Now we want to checkout to the new branch. Use the git command `switch` to change branches.

```
git switch asgn02-inheritance
```

You can work safely on the `asgn02-inheritance` branch. If you totally mess up the code on the `asgn02-inheritance` branch, you can delete the branch without having changed any of your working code on `main`. Here is the code to delete a branch.

```
git branch -d <branch-name>
```

At the end of the assignment, we will merge our new `asgn02-inheritance` branch to our `main` branch.

Class Inheritance

The Challenge

Complete the challenge at the end of chapter 3. 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, or musical instruments 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.

Your Challenge 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. Check 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 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 your are on)
git merge asgn02-inheritance (merges the code from your asgn02-inheritance
git log (see you git history)
```

Create a new branch for the next chapter

```
git switch asgn02-access-control
```

Object Access Control Challenge

Watch the entire Object Access Control tutorial (chapter 4). Use the `challenge-03-oac.php` file. 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 `$wheels` and set values for each class.
- Define a `wheelDetails()` method which returns "it has two wheels" when called.
- Make `weightKg` a private property.
- Define a `setWeightKg()` method (setter method).
- Create a `getter` method to read that value back followed by "kg."
- Modify the `weightLbs()` method to add "lbs" to it.
- What bug have you introduced to the `$weightKg` ?

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

Git

Once finished use git. Same as before.

```
git status (I use git status a lot)
git add . (stage the files)
git commit -m "Completed the access challenge" (commit the files)
git status (same as before)
git switch main (move to the main branch)
git branch (see what branch your are on -- it should be main)
git merge asgn02-access-control (merges the code from your dev branch to yo
git log (see you git history)
```

Switch to the asgn02-access-control branch to continue working git

```
git switch asgn02-access-control
```

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 where necessary. Do not use naive setters and getters. Use **setters** and **getters** to protect your objects if they need validation and you will need to use them if you are referencing a `private` property.

This will seem like overkill for such a small program (it is).

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 switch to the main branch to 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.