Working with Objects.

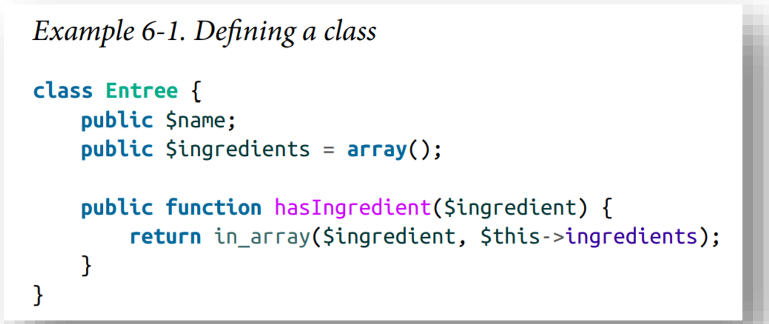
Chapter 6

Object-Oriented Programming

- OOP helps to organize better your code
- Objects are great for making reusable code
- An object is a structure that **combines** data with the logic that operates on it
- Objects provide an organizational structure for grouping related variables and functions together.

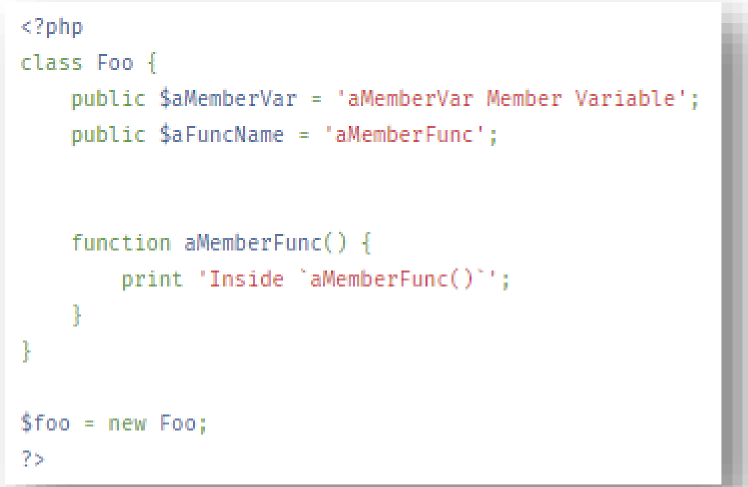
OOP Terms

- **Class:** A template that describes the variables and functions for a kind/group of objects. (e.g. class Entree)
- **Method:** A function defined in a class. (e.g. hasIngredient)
- **Property:** A variable defined in a class. (e.g. $name)
- **Instance:** An individual usage of a class. If you create three instances of a class, each of these instances is based on the same class, they differ by property values. The methods in each instance contain the same instructions, but may produce different results because they have different property values. Creating a new instance of a class is called "instantiating an object."
- **Constructor:** A special method that is automatically run when an object is instantiated. Usually, constructors set up object properties and do other housekeeping that makes the object ready for use.
- **Static method:** A special kind of method that can be called without instantiating a class.



Example 6-1. Defining a class

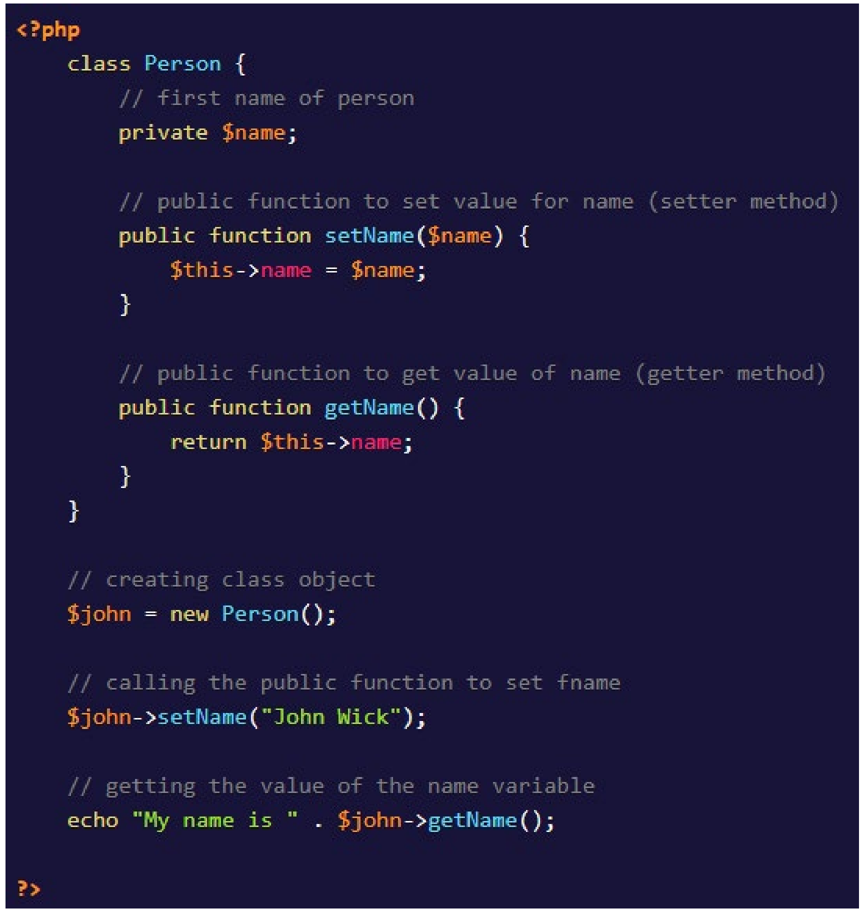
```
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```



```
<?php
class Foo {
    public $aMemberVar = 'aMemberVar Member Variable';
    public $aFuncName = 'aMemberFunc';


    function aMemberFunc() {
        print 'Inside `aMemberFunc()`';
    }
}

$foo = new Foo;
?>
```

this keyword

- this keyword is used inside a class, generally withing the member functions, to access non-static members of a class (variables or functions) for the current object.



```php
<?php
    class Person {
        // first name of person
        private $name;

        // public function to set value for name (setter method)
        public function setName($name) {
            $this->name = $name;
        }

        // public function to get value of name (getter method)
        public function getName() {
            return $this->name;
        }
    }

    // creating class object
    $john = new Person();

    // calling the public function to set fname
    $john->setName("John Wick");

    // getting the value of the name variable
    echo "My name is " . $john->getName();

?>
```

Class and Object

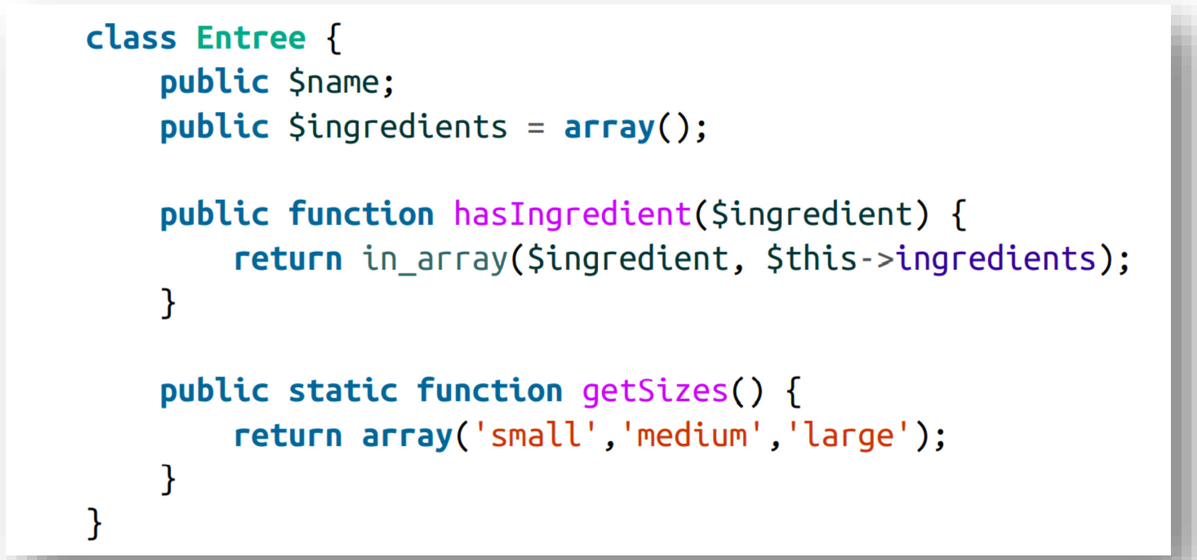
- Defining a class (Entree)
- Then creating two objects/instances (soup and sandwich)
- Use -> to access properties or methods of an object

```php
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

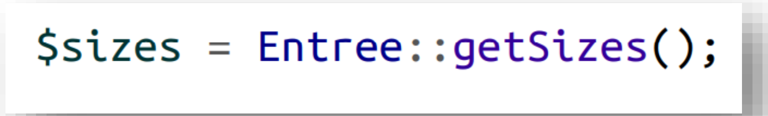
```php
// Create an instance and assign it to $soup
$soup = new Entree;
// Set $soup's properties
$soup->name = 'Chicken Soup';
$soup->ingredients = array('chicken', 'water');

// Create a separate instance and assign it to $sandwich
$sandwich = new Entree;
// Set $sandwich's properties
$sandwich->name = 'Chicken Sandwich';
$sandwich->ingredients = array('chicken', 'bread');


foreach (['chicken','lemon','bread','water'] as $ing) {
    if ($soup->hasIngredient($ing)) {
        print "Soup contains $ing.\n";
    }
    if ($sandwich->hasIngredient($ing)) {
        print "Sandwich contains $ing.\n";
    }
}
```

Static Methods

- Add static keyword before function to make a method static.
- To call a static method, you put :: between the class name and the method name instead of ->



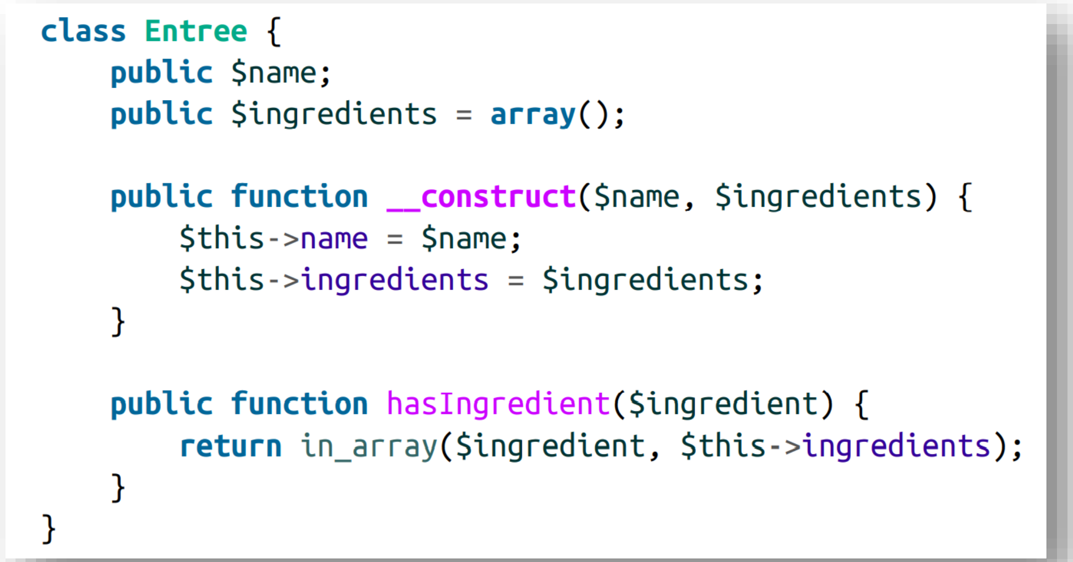
```php
class Entree {
    public $name;
    public $ingredients = array();

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }

    public static function getSizes() {
        return array('small','medium','large');
    }
}
```



```php
$sizes = Entree::getSizes();
```

Constructors

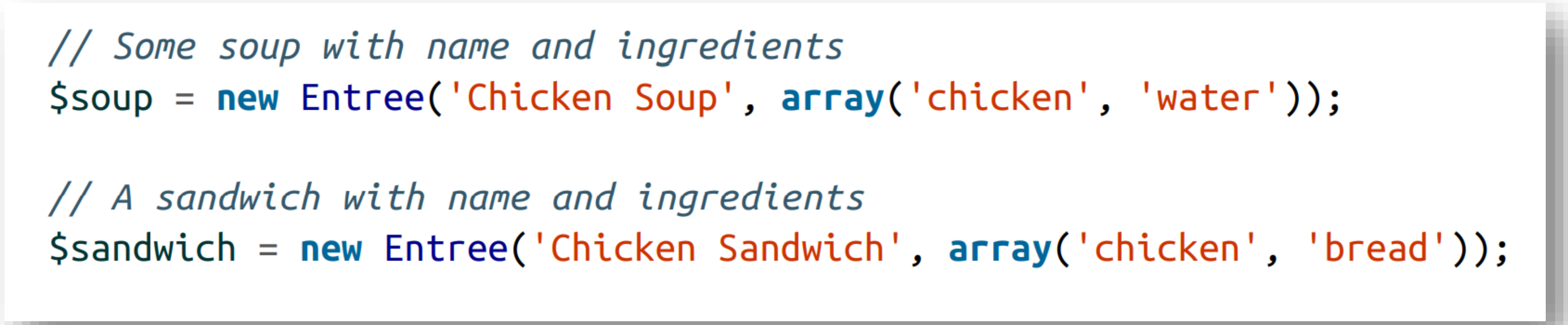
- A constructor is a special method, which is run when the object is created.
- Constructors typically handle setup and housekeeping tasks that make the object ready to use
- In PHP, the constructor method of a class is always called __construct()
- The input parameters can be different name than the properties.
- Inside a constructor, the $this keyword refers to the specific object instance being constructed.
- the constructor function doesn’t return a value



```php
class Entree {
    public $name;
    public $ingredients = array();

    public function __construct($name, $ingredients) {
        $this->name = $name;
        $this->ingredients = $ingredients;
    }

    public function hasIngredient($ingredient) {
        return in_array($ingredient, $this->ingredients);
    }
}
```

Constructors and Creating a New Object

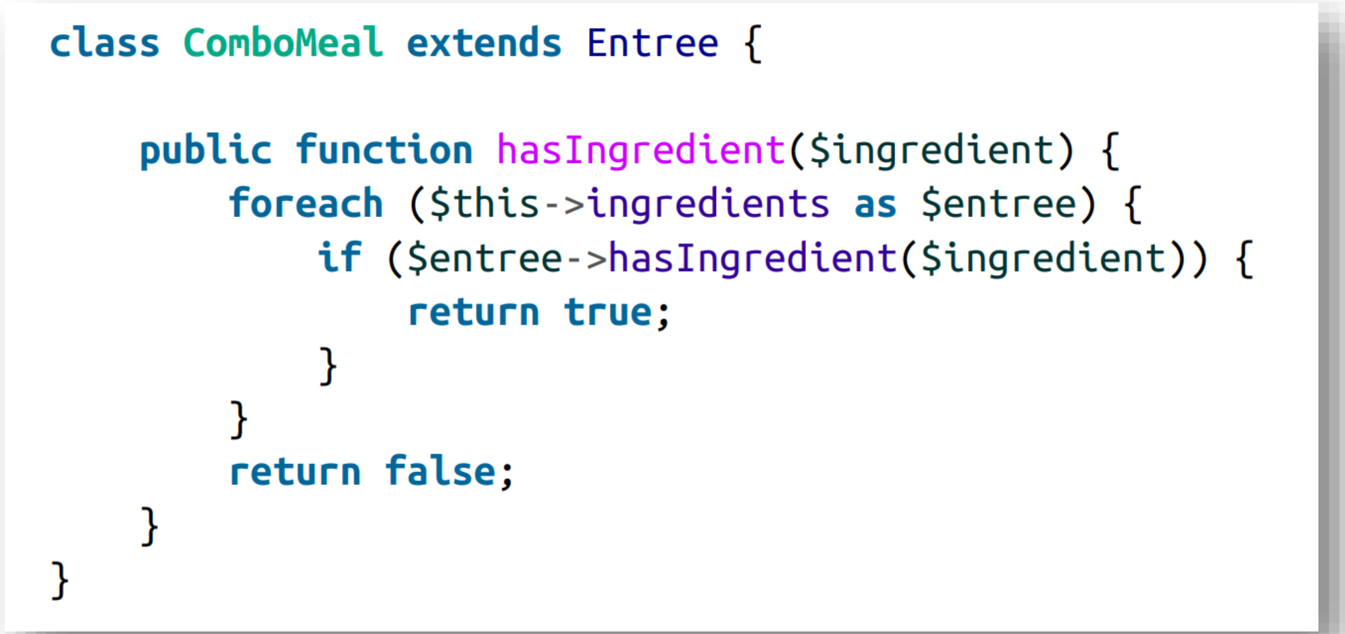
- To pass arguments to the constructor, treat the class name like a function name when you invoke the new operator.



```php
// Some soup with name and ingredients
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// A sandwich with name and ingredients
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));
```

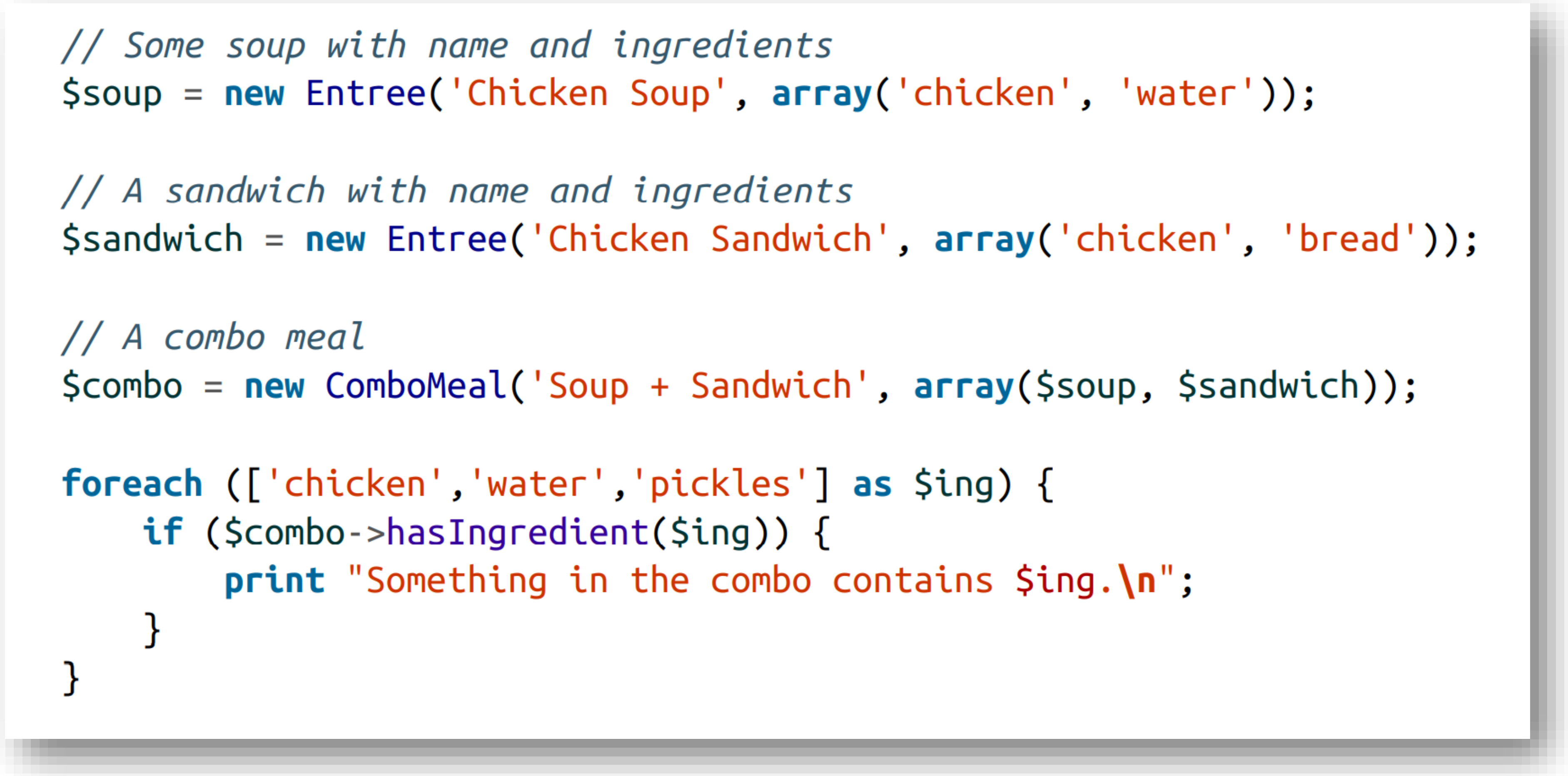
Extending an Object

- A subclass (AKA child class) inherits all the methods and properties of an existing class (the parent class), but then can change them or add its own.
- It's as if you **retyped the definition of Entree** inside the **definition of ComboMeal**, but you get that without actually typing



```
class ComboMeal extends Entree {

    public function hasIngredient($ingredient) {
        foreach ($this->ingredients as $entree) {
            if ($entree->hasIngredient($ingredient)) {
                return true;
            }
        }
        return false;
    }
}
```

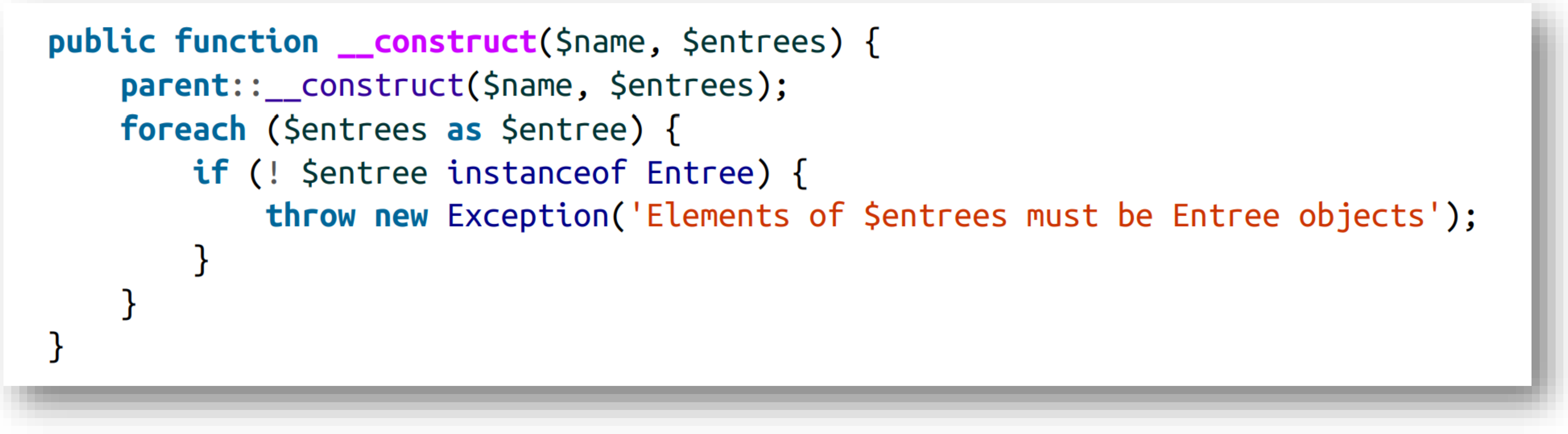
Using a Subclass



```php
// Some soup with name and ingredients
$soup = new Entree('Chicken Soup', array('chicken', 'water'));

// A sandwich with name and ingredients
$sandwich = new Entree('Chicken Sandwich', array('chicken', 'bread'));

// A combo meal
$combo = new ComboMeal('Soup + Sandwich', array($soup, $sandwich));

foreach (['chicken','water','pickles'] as $ing) {
    if ($combo->hasIngredient($ing)) {
        print "Something in the combo contains $ing.\n";
    }
}
```

Constructor for a Subclass

- We can add a constructor for a subclass. We must call the parent constructor **explicitly**
- parent::__construct() refers to the constructor in the parent class.

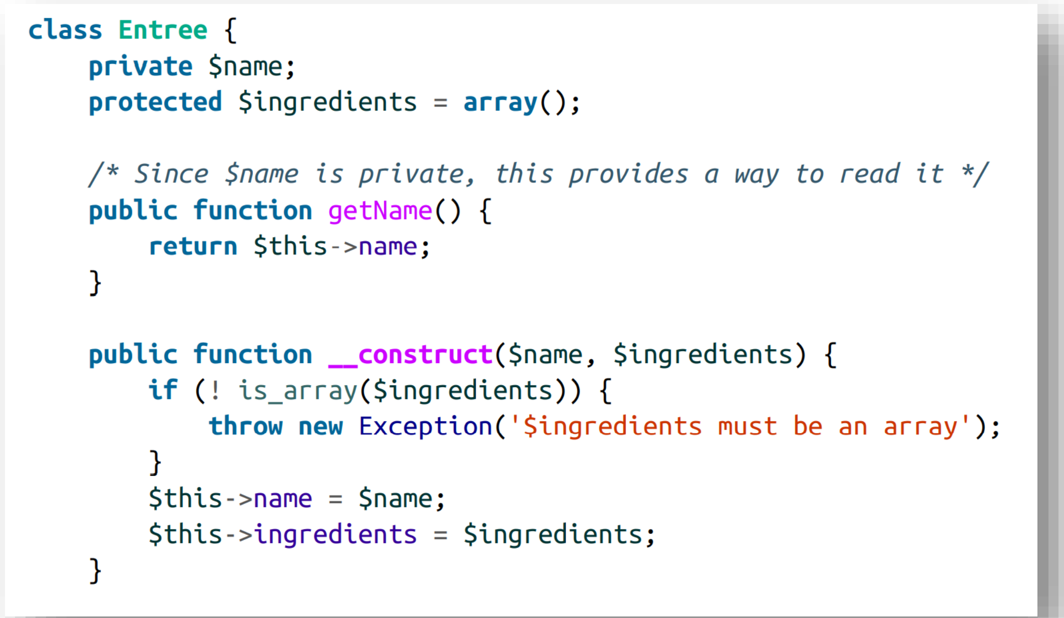


```php
public function __construct($name, $entrees) {
    parent::__construct($name, $entrees);
    foreach ($entrees as $entree) {
        if (! $entree instanceof Entree) {
            throw new Exception('Elements of $entrees must be Entree objects');
        }
    }
}
```

Property and Method Visibility

- Visibility of the properties or methods can be:
  - Public
  - Private
  - Protected

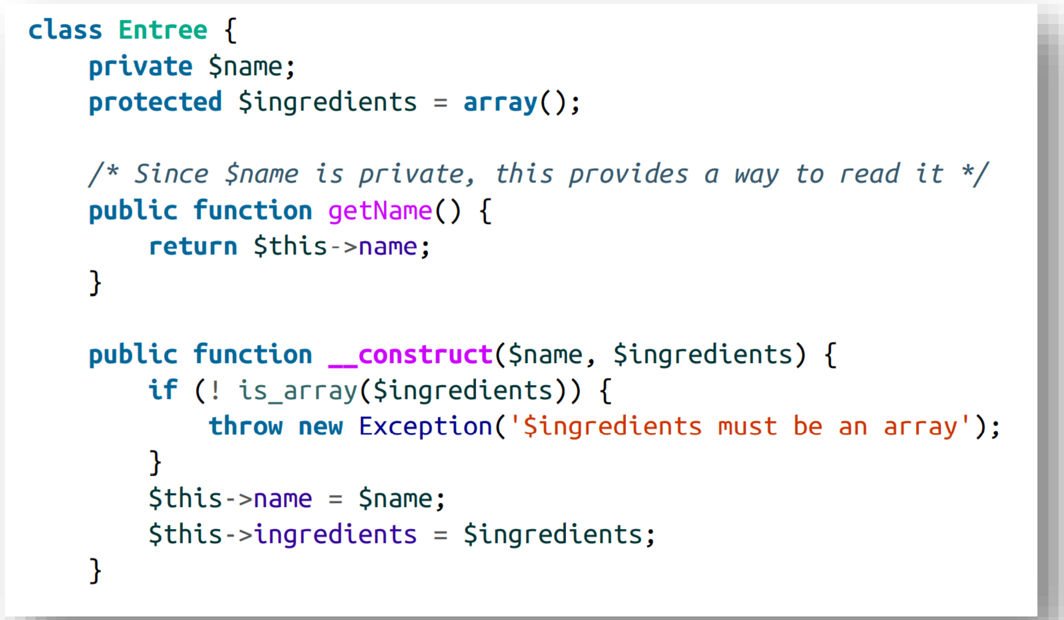
- The public vidibility means all other classes can access the properties and methods.
- The private visibility prevents any code outside the class from accessing the property.
- The protected visibility means that the only subclass code can access the property



```php
class Entree {
    private $name;
    protected $ingredients = array();

    /* Since $name is private, this provides a way to read it */
    public function getName() {
        return $this->name;
    }

    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
            throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }
```

Property and Method Visibility

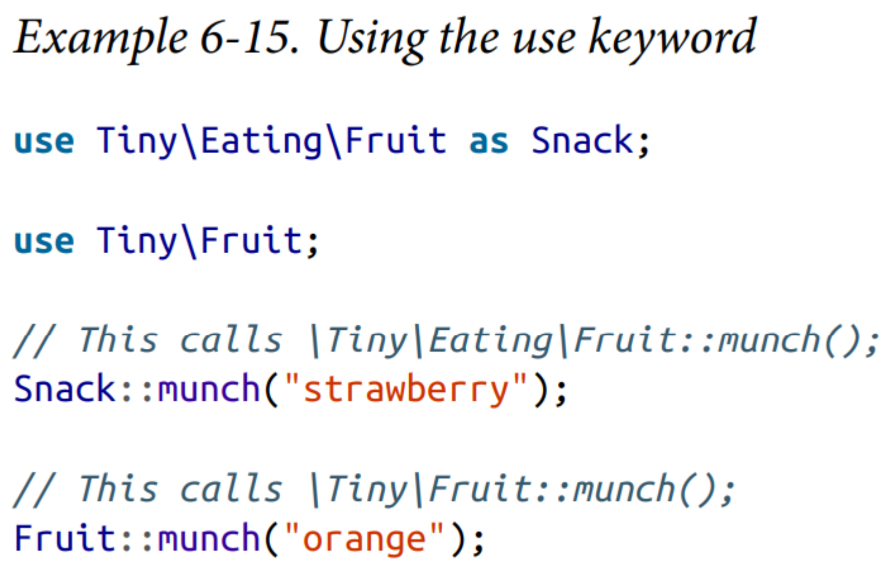
- getName() method is public and provides access to a private property.
- This kind of method is called an accessor. (getter)
- We can have a modifier (setter) method too, to change(set) the value of a private property.



```php
class Entree {
    private $name;
    protected $ingredients = array();

    /* Since $name is private, this provides a way to read it */
    public function getName() {
        return $this->name;
    }

    public function __construct($name, $ingredients) {
        if (! is_array($ingredients)) {
            throw new Exception('$ingredients must be an array');
        }
        $this->name = $name;
        $this->ingredients = $ingredients;
    }
```

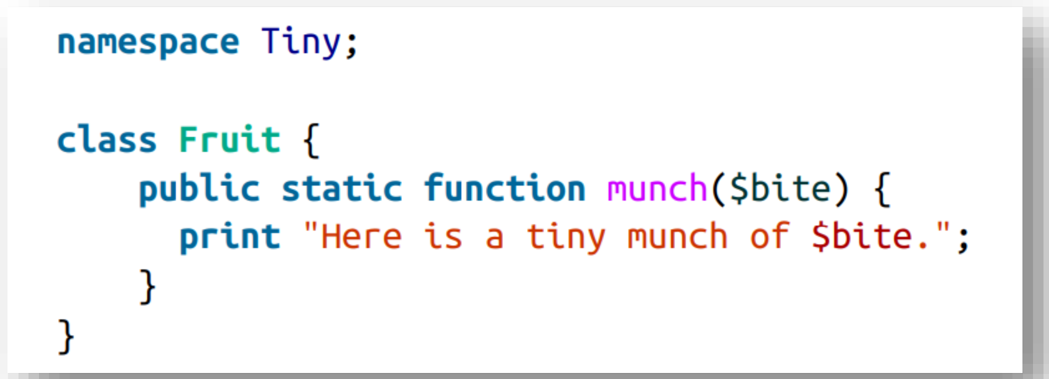
Namespaces

- Namespaces provide:
  - a way to group related code
  - and ensure that names of classes that you've written don't collide with identically named classes written by someone else
- Think of a namespace as a container that can hold class definitions or other namespaces
- To define a class inside a particular namespace, use the namespace keyword at the top of a file with a namespace name.



Example 6-15. Using the use keyword

```
use Tiny\Eating\Fruit as Snack;

use Tiny\Fruit;

// This calls \Tiny\Eating\Fruit::munch();
Snack::munch("strawberry");

// This calls \Tiny\Fruit::munch();
Fruit::munch("orange");
```



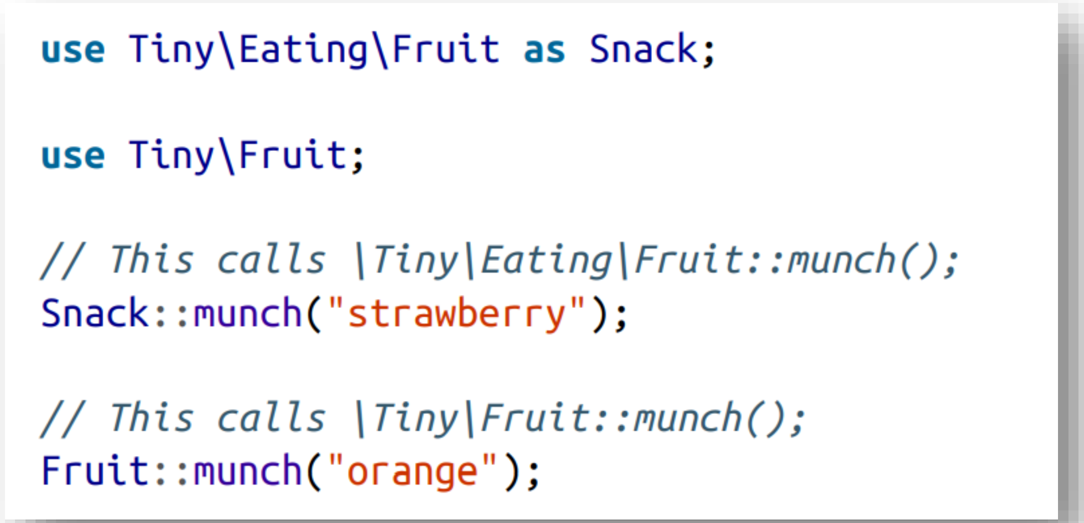
```
namespace Tiny;

class Fruit {
    public static function munch($bite) {
      print "Here is a tiny munch of $bite.";
    }
}
```

Namespaces - use

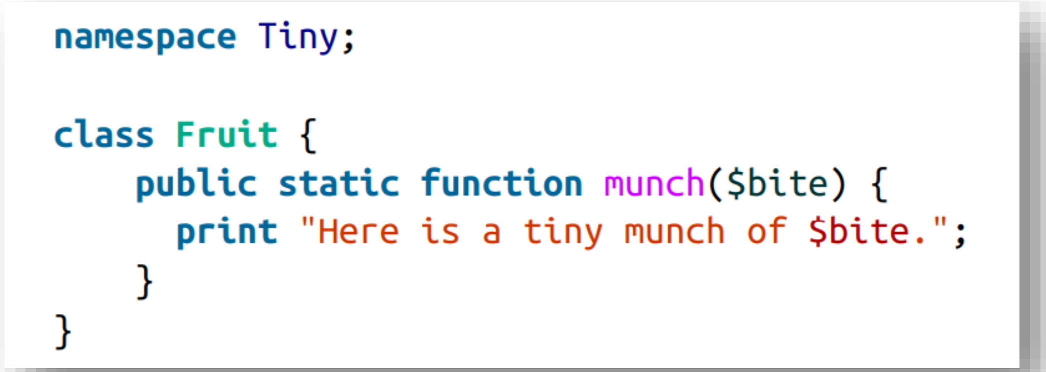
- Now, to invoke munch() on the Fruit class defined you must write:

\Tiny\Fruit::munch("banana");

- You can simplify typing backslashes by using use



```
use Tiny\Eating\Fruit as Snack;

use Tiny\Fruit;

// This calls \Tiny\Eating\Fruit::munch();
Snack::munch("strawberry");

// This calls \Tiny\Fruit::munch();
Fruit::munch("orange");
```



```
namespace Tiny;

class Fruit {
    public static function munch($bite) {
      print "Here is a tiny munch of $bite.";
    }
}
```

Exercises

1. Create a class called `Ingredient`. Each instance of this class represents a single ingredient. The instance should keep track of an ingredient's name and its cost.
2. Add a method to your `IngredientCost` class that changes the cost of an ingredient.
3. Make a subclass of the `Entree` class used in this chapter that accepts `Ingredient` objects instead of string ingredient names to specify the ingredients. Give your `Entree` subclass a method that returns the total cost of the entrée.
4. Put your `Ingredient` class into its own namespace and modify your other code that uses `IngredientCost` to work properly.