Chapter 14

How to create and use objects

Objectives

Applied

1. Create and use your own classes, objects, properties, and methods.
2. Create and use your own class constants, static properties, and static methods.
3. Use your own classes and objects to implement the MVC pattern.

Objectives (continued)

Knowledge

1. Describe the creation of a class including its properties and methods.
2. Describe the use of the $this variable in a class.
3. Describe the creation of an object from a class and the use of the object's properties and methods.
4. Describe the use of class constants and static properties and methods.
5. Describe the differences between applications that use procedural techniques for implementing the MVC pattern and those that use object-oriented techniques.
6. Describe how inheritance works.

The Category class

```
class Category {
    private $id;
    private $name;

    public function __construct($id, $name) {
        $this->id = $id;
        $this->name = $name;
    }
    public function getID() {
        return $this->id;
    }
    public function setID($value) {
        $this->id = $value;
    }
    public function getName() {
        return $this->name;
    }
    public function setName($value) {
        $this->name = $value;
    }
}
```


Key terms

- object-oriented programming
- class
- object
- property
- method

The Product class

```
class Product {
    private $category, $id, $code, $name, $price;

    public function __construct(
            $category, $code, $name, $price) {
        $this->category = $category;
        $this->code = $code;
        $this->name = $name;
        $this->price = $price;
    }
```

The Product class (continued)

```
public function getCategory() {
    return $this->category;
}

public function setCategory($value) {
    $this->category = $value;
}

public function getID() {
    return $this->id;
}

public function setID($value) {
    $this->id = $value;
}
```


The Product class (continued)

```
public function getCode() {
    return $this->code;
}

public function setCode($value) {
    $this->code = $value;
}

public function getName() {
    return $this->name;
}

public function setName($value) {
    $this->name = $value;
}
```


The Product class (continued)

```
public function getPrice() {
    return $this->price;
}

public function getPriceFormatted() {
    $formatted_price =
        number_format($this->price, 2);
    return $formatted_price;
}

public function setPrice($value) {
    $this->price = $value;
}
```


The Product class (continued)

```
public function getDiscountPercent() {
    $discount_percent = 30;
    return $discount_percent;
}

public function getDiscountAmount() {
    $discount_percent =
        $this->getDiscountPercent()/ 100;
    $discount_amount = $this->price * $discount_percent;
    $discount_amount_r = round($discount_amount, 2);
    $discount_amount_f =
    number_format($discount_amount_r, 2);
    return $discount_amount_f;
}
```


The Product class (continued)

```
public function getDiscountPrice() {
    $discount_price =
        $this->price - $this->getDiscountAmount();
    $discount_price_f =
        number_format($discount_price, 2);
    return $discount_price_f;
}
```


The Product class (continued)

```
public function getImageFilename() {
    $image_filename = $this->code . '.png';
    return $image_filename;
}

public function getImagePath() {
    $image_path =
        '../images/' . $this->getImageFilename();
    return $image_path;
}

public function getImageAltText() {
    $image_alt =
        'Image: ' . $this->getImageFilename();
    return $image_alt;
}
}
```

The syntax for coding a property

```
[ public | protected | private ] $propertyName
    [ = initialValue ];
```

A private property

```
private $firstName;
```

A public property with a default value

```
public $comment = '';
```

A protected property

```
protected $counter;
```

Five properties on the same line

```
private $category, $id, $name, $description, $price;
```


Key terms

- public property
- private property
- protected property
- scalar value

The syntax for coding a constructor method

```
public function __construct([parameterList]) {
    // Statements to execute
}
```

The default constructor

```
public function __construct() { }
```

The constructor for the Category class

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

The constructor with default values

```
public function __construct($id = NULL, $name = NULL) {
    $this->id = $id;
    $this->name = $name;
}
```


How to code a destructor method

The syntax

```
public function __destruct() {
    // Statements to execute
}
```

A destructor for a database class

```
public function __destruct() {
    $this->dbConnection->close();
}
```


Key terms

- constructor method
- constructor
- destructor method
- destructor
- object access operator

The syntax for coding a method

```
[public | private | protected]
    function functionName ([parameterList]) {
    // Statements to execute
}
```

A public method

```
public function getSummary() {
    $maxLength = 25;
    $summary = $this->description;
    if (strlen($summary) > $maxLength) {
      $summary =
        substr($summary, 0, $maxLength - 3) . '...';
    }
    return $summary;
}
```

A private method

```
private function internationalizePrice($country = 'US') {
    switch ($country) {
        case 'US':
          return '$' . number_format($this->price, 2);
        case 'FR':
          return number_format(
            $this->price, 2, ',' , '.') . ' EUR';
        default:
          return number_format($this->price, 2);
    }
}
```


A method that accesses a property of the current object

```
public function showDescription() {
    echo $this->description;
}
```

A method that calls a method of the current object

```
public function showPrice($country = 'US') {
    echo $this->internationalizePrice($country);
}
```


The syntax for creating an object

```
$objectName = new ClassName(argumentList);
```

Creating a Category object

```
$brass = new Category(4, 'Brass');
```

Creating a Product object

```
$trumpet = new Product($brass,
                       'Getzen',
                       'Getzen 700SP Trumpet',
                        999.95);
```


The syntax for setting a public property value

```
$objectName->propertyName = value;
```

The syntax for getting a public property value

```
$objectName->propertyName;
```

Setting a property

```
$trumpet->comment = 'Discontinued';
```

Getting a property

```
echo $trumpet->comment;
```


The syntax for calling an object's methods

```
$objectName->methodName(argumentList);
```

Calling the getFormattedPrice() method

```
$price = $trumpet->getFormattedPrice();
```

Object chaining

```
echo $trumpet->getCategory()->getName();
```


Key terms

- instance of a class
- instantiation
- object chaining

A class with three class constants

```
class Person {
    const GREEN_EYES = 1;
    const BLUE_EYES = 2;
    const BROWN_EYES = 3;

    private $eye_color;

    public function getEyeColor() {
        return $this->eye_color;
    }

    public function setEyeColor($value) {
        if ($value == self::GREEN_EYES ||
            $value == self::BLUE_EYES ||
            $value == self::BROWN_EYES) {
            $this->eye_color = $value;
        } else {
            exit('Eye Color Not Set');
        }
    }
}
```


Using a constant outside the class

```
$person = new Person();
$person->setEyeColor(Person::GREEN_EYES);
```


A class with a static property and method

```
class Category {
    private $id, $name;
    private static $objectCount = 0;
    // declare a static property

    public function __construct($id, $name) {
        $this->id = $id;
        $this->name = $name;
        self::$objectCount++;
        // update the static property
    }

    // A public method that gets the static property
    public static function getObjectCount(){
        return self::$objectCount;
    }

    // The rest of the methods for the Category class
}
```


Using a public static method

```
$guitars = new Category(1, 'Guitars');
$basses = new Category(2, 'Basses');
echo '<p>Object count: ' .
    Category::getObjectCount() . '</p>';    // 2

$drums = new Category(3, 'Drums');
echo '<p>Object count: ' .
    Category::getObjectCount() . '</p>';    // 3
```

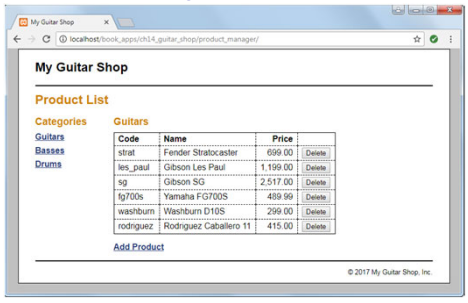
Using a public static property

```
echo '<p>Object count: ' .
    Category::$objectCount . '</p>';
```

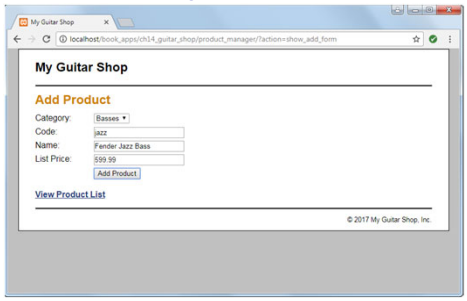

Key terms

- class constant
- scope resolution operator
- static property
- static method
- class property
- class method

The Product List page



The Add Product page



The database.php file

```
<?php
class Database {
    private static $dsn =
        'mysql:host=localhost;dbname=my_guitar_shop1';
    private static $username = 'mgs_user';
    private static $password = 'pa55word';
    private static $db;

    private function __construct() {}
```


The database.php file (continued)

```
    public static function getDB() {
        if (!isset(self::$db)) {
            try {
                self::$db = new PDO(self::$dsn,
                                    self::$username,
                                    self::$password);
            } catch (PDOException $e) {
                $error_message = $e->getMessage();
                include('../errors/database_error.php');
                exit();
            }
        }
        return self::$db;
    }
}
?>
```


The product_db.php file

```
<?php
class ProductDB {
    public static function getProductsByCategory($category_id) {
        $db = Database::getDB();

        $category = CategoryDB::getCategory($category_id);
        $query = 'SELECT * FROM products
                  WHERE products.categoryID = :category_id
                  ORDER BY productID';
        $statement = $db->prepare($query);
        $statement->bindValue(":category_id", $category_id);
        $statement->execute();
        $rows = $statement->fetchAll();
        $statement->closeCursor();
        foreach ($rows as $row) {
            $product = new Product($category,
                                   $row['productCode'],
                                   $row['productName'],
                                   $row['listPrice']);
            $product->setId($row['productID']);
            $products[] = $product;
        }
        return $products;
}
```


The product_db.php file (continued)

```
    public static function getProduct($product_id) {
        $db = Database::getDB();
        $query = 'SELECT * FROM products
                  WHERE productID = :product_id';
        $statement = $db->prepare($query);
        $statement->bindValue(":product_id", $product_id);
        $statement->execute();
        $row = $statement->fetch();
        $statement->closeCursor();

        $category = CategoryDB::getCategory($row['categoryID']);
        $product = new Product($category,
                               $row['productCode'],
                               $row['productName'],
                               $row['listPrice']);
        $product->setID($row['productID']);
        return $product;
    }
```


The product_db.php file (continued)

```
    public static function deleteProduct($product_id) {
        $db = Database::getDB();
        $query = 'DELETE FROM products
                  WHERE productID = :product_id';
        $statement = $db->prepare($query);
        $statement->bindValue(':product_id', $product_id);
        $statement->execute();
        $statement->closeCursor();
    }
```

The product_db.php file (continued)

```
        public static function addProduct($product) {
        $db = Database::getDB();

        $category_id = $product->getCategory()->getID();
        $code = $product->getCode();
        $name = $product->getName();
        $price = $product->getPrice();

        $query = 'INSERT INTO products
                    (categoryID, productCode, productName,
                     listPrice)
                  VALUES
                    (:category_id, :code, :name, :price)';
        $statement = $db->prepare($query);
        $statement->bindValue(':category_id', $category_id);
        $statement->bindValue(':code', $code);
        $statement->bindValue(':name', $name);
        $statement->bindValue(':price', $price);
        $statement->execute();
        $statement->closeCursor();
    }
}
?>
```


The index.php file

```
<?php
require('../model/database.php');
require('../model/category.php');
require('../model/category_db.php');
require('../model/product.php');
require('../model/product_db.php');

$action = filter_input(INPUT_POST, 'action');
if ($action == NULL) {
    $action = filter_input(INPUT_GET, 'action');
    if ($action == NULL) {
        $action = 'list_products';
    }
}
```


The index.php file (continued)

```
if ($action == 'list_products') {
    $category_id = filter_input(INPUT_GET, 'category_id',
            FILTER_VALIDATE_INT);
    if ($category_id == NULL || $category_id == FALSE) {
        $category_id = 1;
    }
    $current_category = CategoryDB::getCategory($category_id);
    $categories = CategoryDB::getCategories();
    $products = ProductDB::getProductsByCategory($category_id);
    include('product_list.php');
} else if ($action == 'delete_product') {
    $product_id = filter_input(INPUT_POST, 'product_id',
            FILTER_VALIDATE_INT);
    $category_id = filter_input(INPUT_POST, 'category_id',
            FILTER_VALIDATE_INT);
    ProductDB::deleteProduct($product_id);
    header("Location: .?category_id=$category_id");
} else if ($action == 'show_add_form') {
    $categories = CategoryDB::getCategories();
    include('product_add.php');
```


The index.php file (continued)

```
} else if ($action == 'add_product') {
    $category_id = filter_input(INPUT_POST, 'category_id',
            FILTER_VALIDATE_INT);
    $code = filter_input(INPUT_POST, 'code');
    $name = filter_input(INPUT_POST, 'name');
    $price = filter_input(INPUT_POST, 'price');
    if ($category_id == NULL || $category_id == FALSE ||
        $code == NULL ||
            $name == NULL || $price == NULL || $price == FALSE) {
        $error = "Invalid product data. Check all fields and try
                  again.";
        include('../errors/error.php');
    } else {
        $category = CategoryDB::getCategory($category_id);
        $product = new Product($category, $code, $name, $price);
        ProductDB::addProduct($product);
        header("Location: .?category_id=$category_id");
    }
}
?>
```


The product_list.php file

```
<?php include '../view/header.php'; ?>
<main>
    <h1>Product List</h1>
    <aside>
        <!-- display a list of categories -->
        <h2>Categories</h2>
        <nav>
        <ul>
        <?php foreach ($categories as $category) : ?>
            <li>
            <a href="?category_id=<?php echo $category->getID();
            ?>">
                <?php echo $category->getName(); ?>
            </a>
            </li>
        <?php endforeach; ?>
        </ul>
        </nav>
    </aside>
```


The product_list.php file (continued)

```
    <section>
        <!-- display a table of products -->
        <h2><?php echo $current_category->getName(); ?></h2>
        <table>
            <tr>
                <th>Code</th>
                <th>Name</th>
                <th class="right">Price</th>
                <th> </th>
            </tr>
```

The product_list.php file (continued)

```
        <?php foreach ($products as $product) : ?>
        <tr>
            <td><?php echo $product->getCode(); ?></td>
            <td><?php echo $product->getName(); ?></td>
            <td class="right"><?php echo
                $product->getPriceFormatted(); ?>
            </td>
            <td><form action="." method="post"
                      id="delete_product_form">
                <input type="hidden" name="action"
                       value="delete_product">
                <input type="hidden" name="product_id"
                       value="<?php echo $product->getID(); ?>">
                <input type="hidden" name="category_id"
                       value="<?php echo
                    $current_category->getID(); ?>">
                <input type="submit" value="Delete">
            </form></td>
        </tr>
        <?php endforeach; ?>
    </table>
```


The product_list.php file (continued)

```
        <p><a href="?action=show_add_form">Add Product</a></p>
    </section>
</main>
<?php include '../view/footer.php'; ?>
```


The syntax for looping through an object's properties

```
foreach($objectName as
        [ $propertyName => ] $propertyValue) {
    // statements to execute
}
```


An Employee class

```
class Employee {
    public $firstName, $lastName;
    private $ssn, $dob;

    public function __construct($first, $last) {
        $this->firstName = $first;
        $this->lastName = $last;
    }

    // getSSN, setSSN, getDOB, setDOB methods not shown

    // Show properties - private, protected, and public
    public function showAll() {
        echo '<ul>';
        foreach($this as $name => $value ) {
            echo "<li>$name = $value</li>";
        }
        echo '</ul>';
    }
}
```


An Employee object with four properties

```
$employee = new Employee('John', 'Doe');
$employee->setSSN('999-14-3456');
$employee->setDOB('3-15-1970');
```

Show all properties

```
$employee->showAll();
```

Show public properties only

```
echo '<ul>';
foreach($employee as $name => $value ) {
    echo "<li>$name = $value</li>";
}
echo '</ul>';
```


The syntax for cloning an object

```
clone $objectName
```

An object to clone

```
$brass = new Category(4, 'Brass');
$trumpet = new Product($brass,
    'Getzen', 'Getzen 700SP Trumpet', 999.95);
```

Create a second reference to the object

```
$trombone = $trumpet;
// both variables refer to the same object

$trombone->setPrice(699.95);
// changes the price for both variables
```

Create a clone of the object

```
$trombone = clone $trumpet;
// copy the object

$trombone->setPrice(899.95);
// only changes the price for trombone
```

The copies are shallow copies

```
$trombone->getCategory()->setName('Orchestral Brass');
echo $trumpet->getCategory()->getName();
// Displays 'Orchestral Brass'
```


Using the == operator to compare objects

```
$result_1 = ($trumpet == $trombone);
// $result_1 is FALSE

$flugelhorn = clone $trumpet;
$result_2 = ($trumpet == $flugelhorn);
// $result_2 is TRUE
```

Using the === operator to compare objects

```
$result_3 = ($trumpet === $flugelhorn);
// $result_3 is FALSE

$trumpet_2 = $trumpet;
$result_4 = ($trumpet === $trumpet_2);
// $result_4 is TRUE

$result_5 = ($trumpet->getCategory() ===
             $trombone->getCategory());
// $result_5 is TRUE
```


Key terms

- clone
- shallow copy

Functions for inspecting an object

```
class_exists($class)
get_class($object)
is_a($object, $class)
property_exists($object, $property)
method_exists($object, $method)
```


Determine if an object is an instance of a class

```
if (is_a($trumpet, 'Product')) {
    // Code to work with a Product object
}
```

Determine if an object has a property

```
if (property_exists($trumpet, 'price')) {
    // Code to work with the price property
}
```

Determine if an object has a method

```
if (method_exists($trumpet, 'getPrice')) {
    // Code to work with the getPrice method
}
```


Key terms

- introspection
- reflection

A superclass

```
class Person {
    private $firstName, $lastName, $phone, $email;

    public function __construct($first, $last) {
        $this->firstName = $first;
        $this->lastName  = $last;
    }

    public function getFirstName()
        { return $this->firstName;   }
    public function setFirstName($value)
        { $this->firstName = $value; }
    public function getLastName()
        { return $this->lastName;   }
    public function setLastName($value)
        { $this->lastName = $value; }
    public function getPhone()
        { return $this->phone;   }
    public function setPhone($value)
        { $this->phone = $value; }
```


A superclass (continued)

```
    public function getEmail()
        { return $this->email;   }
    public function setEmail($value)
        { $this->email = $value; }
}
```


A subclass

```
class Employee extends Person {
    private $ssn, $hireDate;

    public function __construct(
            $first, $last, $ssn, $hireDate) {
        $this->ssn = $ssn;
        $this->hireDate = $hireDate;

        // Finish initialization
        parent::__construct($first, $last);
    }

    public function getSSN()
        { return $this->ssn;   }
    public function setSSN($value)
        { $this->ssn = $value; }
    public function getHireDate()
        { return $this->hireDate;   }
    public function setHireDate($value)
        { $this->hireDate = $value; }
}
```


Code that uses the subclass

```
$emp = new Employee(
    'John', 'Doe', '999-14-3456', '8-25-1996');
$emp->setPhone('919-555-4321');
// Inherited from Person Class
```


Key terms

- inheritance
- inherit
- subclass, derived class, child class
- superclass, base class, parent class
- extend a superclass
- override a method of a superclass

How the access modifiers work

Modifier	Access outside class?	Access from subclass?
public	Yes	Yes
protected	No	Yes
private	No	No

A superclass

```
class Person {
    protected $firstName, $lastName;
    private   $phone, $email;

    // get and set methods
}
```

A subclass

```
class Employee extends Person {
    private $ssn, $hireDate;

    // get and set methods

    // Using the protected properties
    // from the Person class
    public function getFullName() {
        return $this->lastName . ', ' .
               $this->firstName;
    }
}
```


An abstract class with an abstract method

```
abstract class Person {
    private $firstName, $lastName, $phone, $email;

    // get and set methods

    // An abstract method
    abstract public function getFullName();
}
```


Implementing an abstract class

```
class Customer extends Person {
    private $cardNumber, $cardType;

    public function __construct(
            $first, $last, $phone, $email) {
        $this->setPhone($phone);
        $this->setEmail($email);
        parent::__construct($first, $last);
    }

    public function getCardNumber()
        { return $this->cardNumber;   }
    public function setCardNumber($value)
        { $this->cardNumber = $value; }
    public function getCardType()
        { return $this->cardType;   }
    public function setCardType($value)
        { $this->cardType = $value; }
```


Implementing an abstract (continued)

```
    // Concrete implementation of the abstract method
    public function getFullName() {
        return $this->getFirstName() . ' ' .
               $this->getLastName();
    }
}
```


Code that attempts to create an object from the abstract class

```
$customer = new Person('John', 'Doe');
// Fatal error
```

Code that creates and uses an object from the concrete class

```
$customer = new Customer(
    'John', 'Doe', '919-555-4321', 'jdoe@example.com');
echo '<p>' . $customer->getFullName() . '</p>';
```


Key terms

- abstract class
- abstract method
- concrete class

A class with a final method

```
class Person {
    // Other properties and methods not shown here

    final public function getFirstName() {
        return $this->firstName;
    }
}
```

A subclass that tries to override the final method

```
class Employee extends Person {
    // Other properties and methods not shown here

    // Attempt to override a final method - fatal error
    public function getFirstName() {
        return ucwords($this->firstName);
    }
}
```


A final class that can't be inherited

```
final class Employee extends Person {
    // Properties and methods for class
}
```

A class that attempts to inherit the final class

```
class PartTime extends Employee {
    // Properties and methods for class
}
```


The syntax for creating an interface

```
interface interfaceName {
    const constantName = constantValue;
    public function methodName( parameterList );
}
```

An interface to show an object

```
interface Showable {
    public function show();
}
```

An interface that requires two test methods

```
interface Testable {
    public function test1($value1);
    public function test2($value1, $value2);
}
```


An interface that provides three constants

```
interface EyeColor {
    const GREEN = 1;
    const BLUE = 2;
    const BROWN = 3;
}
```


A class that inherits a class and implements an interface

```
class Employee extends Person implements Showable {
    // get and set methods

    // Implement the Showable interface
    public function show() {
        echo 'First Name: ' .
            $this->getFirstName() . '<br>';
        echo 'Last Name: ' .
            $this->getLastName() . '<br>';
        echo 'SSN: ' .
            $this->ssn . '<br>';
        echo 'Hire Date: ' .
            $this->hireDate . '<br>';
    }
}
```


A class declaration that implements three interfaces

```
class Customer extends Person
              implements Showable,
              Testable,
              EyeColor { ... }
```

Key terms

- interface
- implementing an interface