Session 14

How to create and use objects

Objectives

- How to create and use classes
- How to code class constants, properties, and methods
- The object-oriented Product manager application
- Additional skills for working with object
- How to work with inheritance

How to create and use classes

The code for the Category class

- Object-oriented programming groups related variables and functions into data structures called objects
- A class defines the properties and methods of a particular type of object.

The code for the Category class (cont.)

- 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;
    }
}
```

The code for the Product class

- 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;
    }

    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;
    }

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

The code for the Product class (cont.)

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

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

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

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 code for the Product class (cont.)

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

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

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

The code for the Product class (cont.)

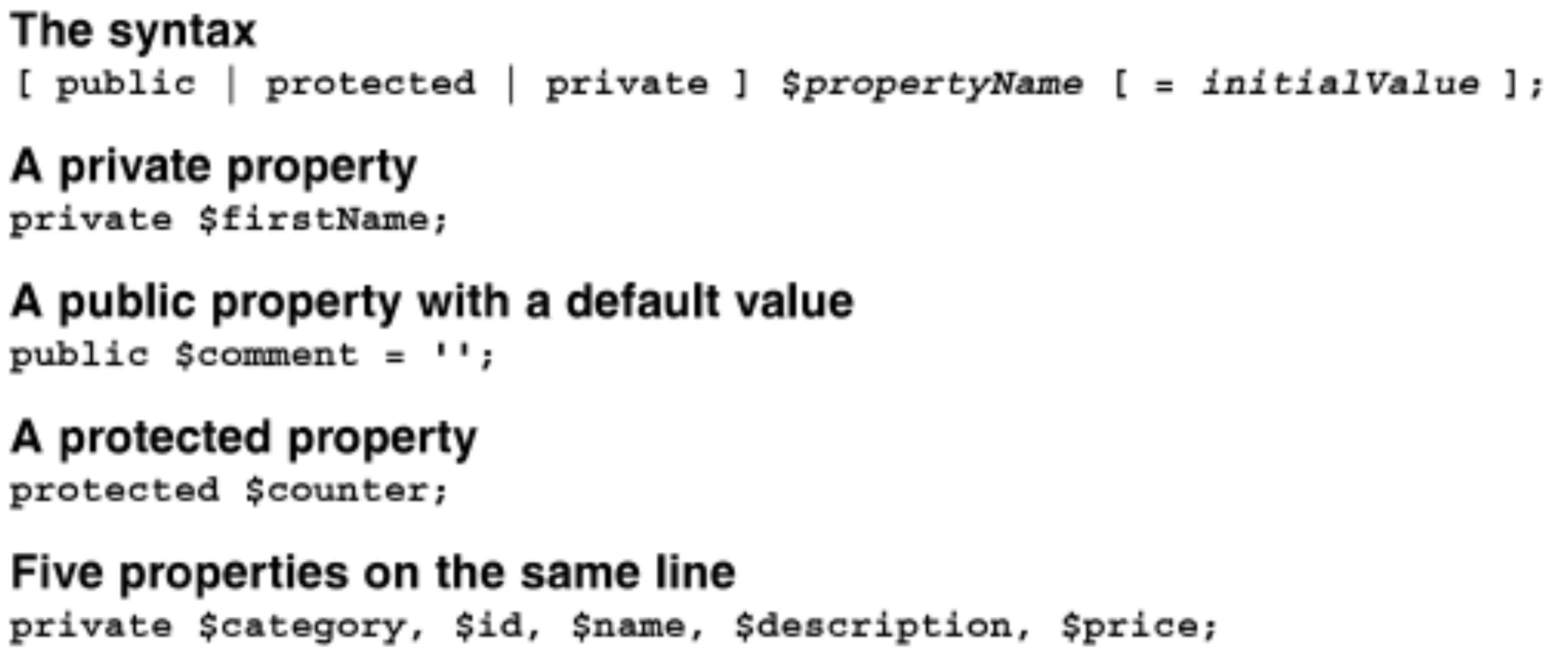
```
    public function getImageFilename() {
        $image_filename = $this->code . '.png';
        return $image_filename;
    }

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

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

How to code properties

- A public properties can be directly accessed by code outside of the class
- Private and protected properties can’t be directly accessed by code outside the class

How to code properties (cont.)

- How to code a property



The syntax

```
[ 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;
```

How to code constructors and destructors

- A constructor method, or just constructor, is a special method that is executed when a new object is created from the class. It often initializes the properties of the object.
- A destructor method, or just destructor, is a special method that’s executed when an object is no longer available for use. In other words, it is executed when there are no variables that refer to the object.

How to code constructors and destructors (cont.)

- Within a class, the special variable named $this stores a reference to the current object
- The object access operator (->) provides access to an object’s properties and methods.
- How to code a constructor method
  - The syntax

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

How to code constructors and destructors (cont.)

– 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 for the Category class with default value

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

How to code constructors and destructors (cont.)

- 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();
}
```

How to code methods

- How to code a method
  - The syntax

```
[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;
}
```

How to code methods (cont.)

– A private method

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

– A method that accesses a property of the current object

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

How to code methods (cont.)

– A method that calls a method of the current object

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

How to create and use object

- An object is an instance of a class.
- The process of creating an object from a class is some time called instantiation.
- To access an object’s property, you code a reference to the object followed by the object access operator (->) and the name property.
- To call an object’s method, you code a reference to the object followed by the object access operator (->), the name of method, and set of parentheses.

How to create and use object (cont.)

- How to create an object
  - The syntax

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

  - Create a Category object

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

  - Create a Product object.

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

How to create and use object (cont.)

- How to access an object's properties
  - The syntax for setting a public property value

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

  - The syntax for getting a public property value

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

  - Set a property

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

  - Get a property

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

How to create and use object (cont.)

- How to call an object's methods
  - The syntax

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

  - Call the getFormattedPrice method

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

  - Object chaining

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

How to code class constants, properties, and methods

How to code class constants

- A class constant is a constant value that belongs to the class, not object created from the class
- To access a constant that belongs to a class, you can code the name of the class followed by a double colon.
- Inside a class, you can access a class constant by coding the self keyword followed by a double colon and the class constant name.
- Outside class, you can access a class constant by coding the class name followed by a double colon and the class constant name.

How to code class constants (cont.)

- How to create a class constant

```
class Person {
    const MALE = 'm';
    const FEMALE = 'f';

    private $gender;

    public function getGender() {
        return $this->gender;
    }

    public function setGender($value) {
        if ($value == self::MALE || $value == self::FEMALE) {
            $this->gender = $value;
        } else {
            exit('Invalid Gender');
        }
    }
}
```

How to code class constants (cont.)

– Use the constant outside the class

```
$person = new Person();
$person->setGender(Person::FEMALE);
```

How to code static properties and methods

- A static property or static method is a property or method that belongs to a class, not to objects created from the class.
- Inside class, you can access a static property or method by coding the self keyword followed by a double colon and the property or method name
- Outside a class, you can access a static property or method that’s public by coding the class name followed by a double colon and the static property or method name

How to code static properties and methods (cont.)

- How to create static properties and methods
  - 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
}
```

How to code static properties and methods (cont.)

– Using a static method

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

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

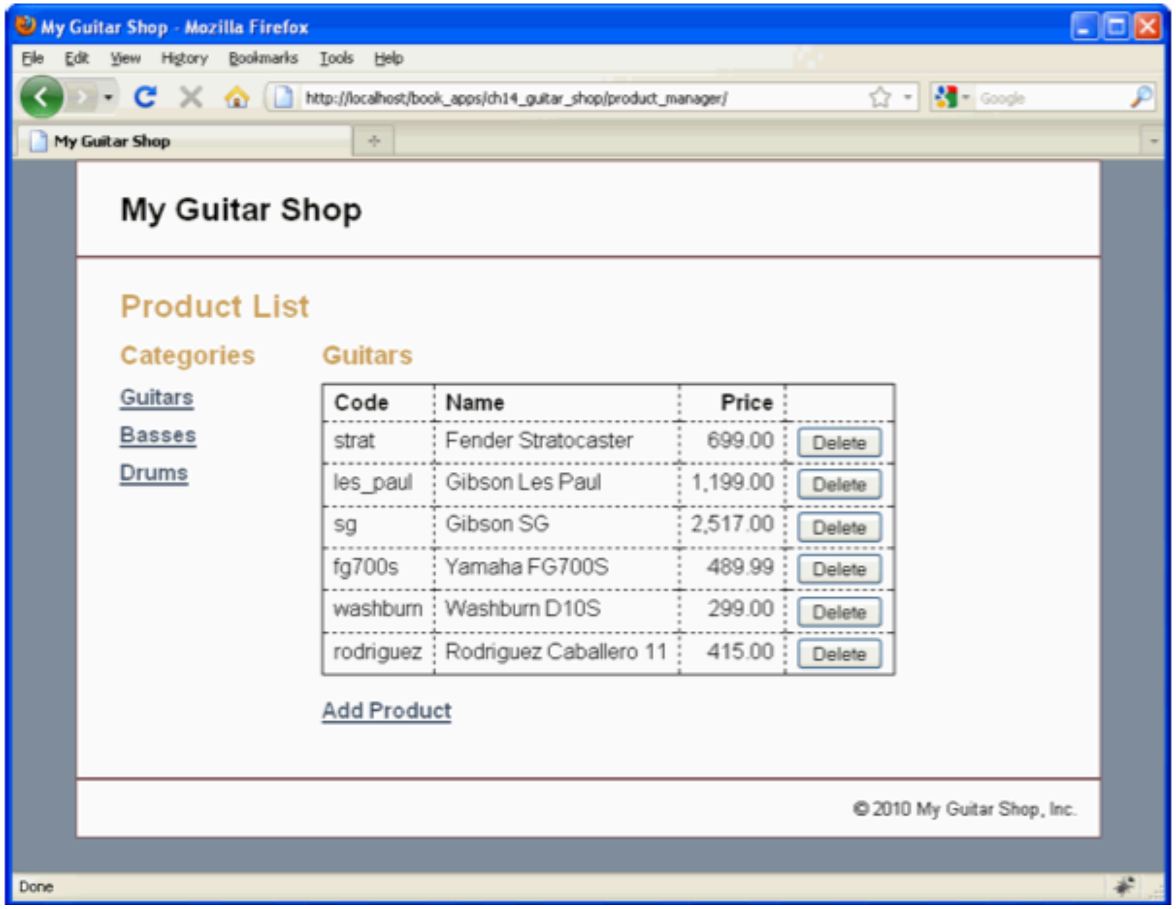
– Using a public static property

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

The object-oriented Product Manager application

The user interface

- The Product List page


My Guitar Shop - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/book_apps/ch14_guitar_shop/product_manager/
My Guitar Shop
Done

My Guitar Shop

Product List

Categories

Guitars

Basses

Drums

Guitars

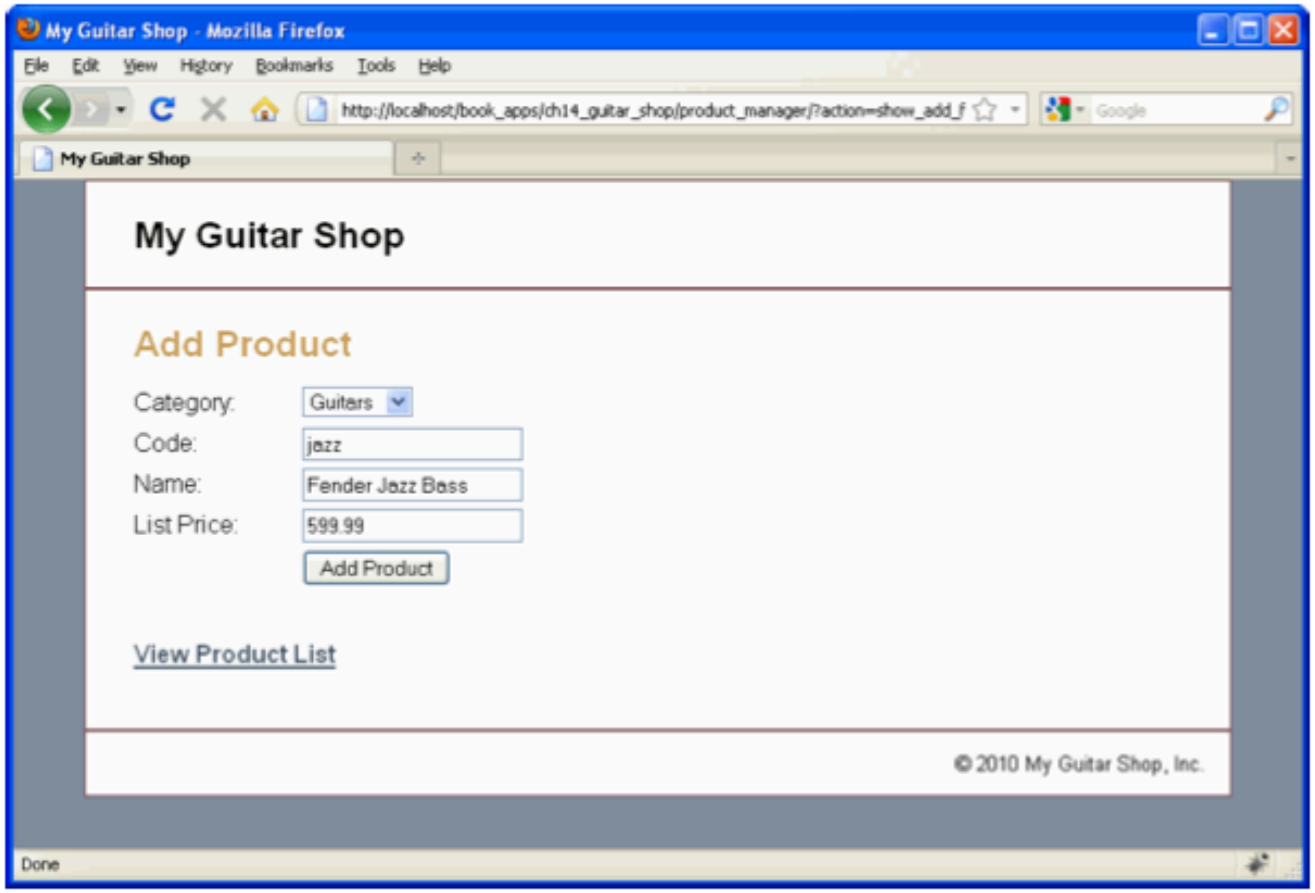
Code	Name	Price	
strat	Fender Stratocaster	699.00	Delete
les_paul	Gibson Les Paul	1,199.00	Delete
sg	Gibson SG	2,517.00	Delete
fg700s	Yamaha FG700S	489.99	Delete
washburn	Washburn D10S	299.00	Delete
rodriguez	Rodriguez Caballero 11	415.00	Delete

Add Product

© 2010 My Guitar Shop, Inc.

The user interface (cont.)

- The Add Product page


My Guitar Shop - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost/book_apps/ch14_guitar_shop/product_manager/?action=show_add_f
Google
My Guitar Shop
Done

My Guitar Shop

Add Product

Category:	Guitars
Code:	jazz
Name:	Fender Jazz Bass
List Price:	599.99

Add Product

View Product List

© 2010 My Guitar Shop, Inc.

The model

- The database.php file

```php
<?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() {}

    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 model (cont.)

- The product_db.php file

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

        $category = CategoryDB::getCategory($category_id);

        $query = "SELECT * FROM products
                  WHERE categoryID = '$category_id'
                  ORDER BY productID";
        $result = $db->query($query);
        $products = array();
        foreach ($result as $row) {
            $product = new Product($category,
                                   $row['productCode'],
                                   $row['productName'],
                                   $row['listPrice']);
            $product->setID($row['productID']);
            $products[] = $product;
        }
        return $products;
    }
```

The model (cont.)

```
public static function getProduct($product_id) {
    $db = Database::getDB();
    $query = "SELECT * FROM products
              WHERE productID = '$product_id'";
    $result = $db->query($query);
    $row = $result->fetch();
    $category = CategoryDB::getCategory($row['categoryID']);
    $product = new Product($category,
                           $row['productCode'],
                           $row['productName'],
                           $row['listPrice']);
    $product->setID($row['productID']);
    return $product;
}

public static function deleteProduct($product_id) {
    $db = Database::getDB();
    $query = "DELETE FROM products
              WHERE productID = '$product_id'";
    $row_count = $db->exec($query);
    return $row_count;
}
```

The model (cont.)

```
    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')";

        $row_count = $db->exec($query);
        return $row_count;
    }
}
?>
```

The controller

- 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');

if (isset($_POST['action'])) {
    $action = $_POST['action'];
} else if (isset($_GET['action'])) {
    $action = $_GET['action'];
} else {
    $action = 'list_products';
}

if ($action == 'list_products') {
    $category_id = $_GET['category_id'];
    if (!isset($category_id)) {
        $category_id = 1;
    }

    $current_category = CategoryDB::getCategory($category_id);
    $categories = CategoryDB::getCategories();
    $products = ProductDB::getProductsByCategory($category_id);
```

The controller (cont.)

```
    include('product_list.php');
} else if ($action == 'delete_product') {
    $product_id = $_POST['product_id'];
    $category_id = $_POST['category_id'];

    ProductDB::deleteProduct($product_id);

    header("Location: .?category_id=$category_id");
} else if ($action == 'show_add_form') {
    $categories = CategoryDB::getCategories();
    include('product_add.php');
} else if ($action == 'add_product') {
    $category_id = $_POST['category_id'];
    $code = $_POST['code'];
    $name = $_POST['name'];
    $price = $_POST['price'];

    if (empty($code) || empty($name) || empty($price)) {
        $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 view

- The product_list.php file

```
<?php include '../view/header.php'; ?>
<div id="main">

    <h1>Product List</h1>

    <div id="sidebar">
        <!-- display a list of categories -->
        <h2>Categories</h2>
        <ul class="nav">
        <?php foreach ($categories as $category) : ?>
            <li>
            <a href="?category_id=<?php echo $category->getID(); ?>">
                <?php echo $category->getName(); ?>
            </a>
            </li>
        <?php endforeach; ?>
        </ul>
    </div>
```

The view (cont.)

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

The view (cont.)

```
                    <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>
        <p><a href="?action=show_add_form">Add Product</a></p>
    </div>

</div>
<?php include '../view/footer.php'; ?>
```

Additional skills for working with objects

How to loop through an object’s properties

- You can use a foreach loop to access each property in an object
- A foreach loop coded inside a method of an object loops through the object’s private, protected, and public properties
- A foreach loop coded outside an object only loops through the object’s public properties

How to loop through an object's properties (cont.)

- How to loop through an object's properties
  - the syntax

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

How to loop through an object’s properties (cont.)

– Define an Employee class

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

How to loop through an object’s properties (cont.)

- Create an Employee object with four properties

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

- Sow all properties

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

- Show public properties only

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

How to clone and compare objects

- To create a second reference to an object, you can use the equals (=) operator. To create a copy, or clone, of an object, you must use the clone operator.
- How to clone an object
  - The syntax

```
clone $objectName
```

  - An object to clone

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

How to clone and compare objects (cont.)

- Create a second reference to an object

```
$trombone = $trumpet;          // both variables refer to the same object
$trombone->setPrice(699.95);   // changes the price for both variables
```

- Create a clone of an object

```
$trombone = clone $trumpet;    // copy the object
$trombone->setPrice(899.95);   // this only changes the price for trombone
```

- The copies are shallow copies

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

How to clone and compare objects (cont.)

- Compare object
  - Use the equality operator (==) to check whether both objects are instances of the same class and have same values for every property
  - Use the identity operator (===) to check whether both object variables refer to the same instance of an object
  - Use the logical not versions of the equality (!=) and identity (!==) operators

How to clone and compare objects (cont.)

- How to compare two objects
  - Use the equality (==) operator

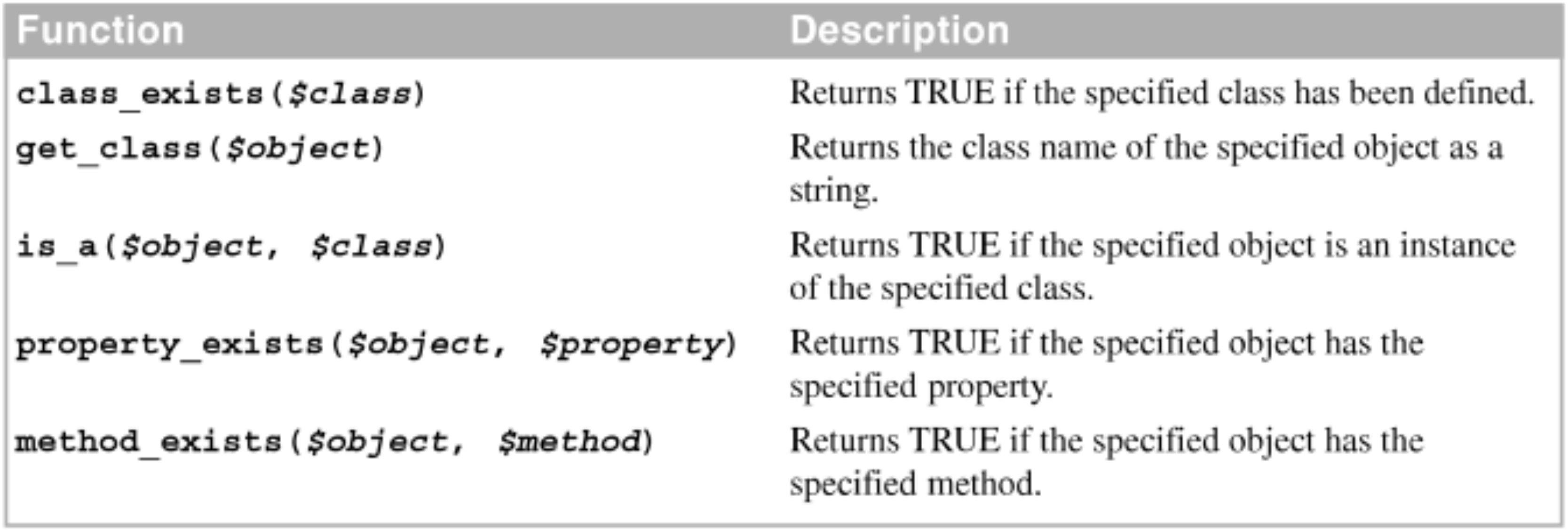
```
$result_1 = ($trumpet == $trombone);      // $result_1 is FALSE

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

  - Use the identity (===) operator

```
$result_3 = ($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
```

How to inspect an object

- Inspecting an object is known as introspection or reflection
- The URL for the reflection API
- Functions for inspecting an object



Function	Description
`class_exists($class)`	Returns TRUE if the specified class has been defined.
`get_class($object)`	Returns the class name of the specified object as a string.
`is_a($object, $class)`	Returns TRUE if the specified object is an instance of the specified class.
`property_exists($object, $property)`	Returns TRUE if the specified object has the specified property.
`method_exists($object, $method)`	Returns TRUE if the specified object has the specified method.

How to inspect an object (cont.)

- 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
}
```

How to work with inheritance

How to inherit a class

- Inheritance provides a way to create a new class based on an existing class. The new class inherits the properties and methods of the existing class
- A class that inherits from a class is called a subclass, derived class, or child class. A class that is inherited by another class is called a superclass, base class or parent class.
- A subclass can extend the superclass by adding new properties and methods.

How to inherit a class (cont.)

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

How to inherit a class (cont.)

- A subclass

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

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

        // Call Person constructor to 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; }
}
```

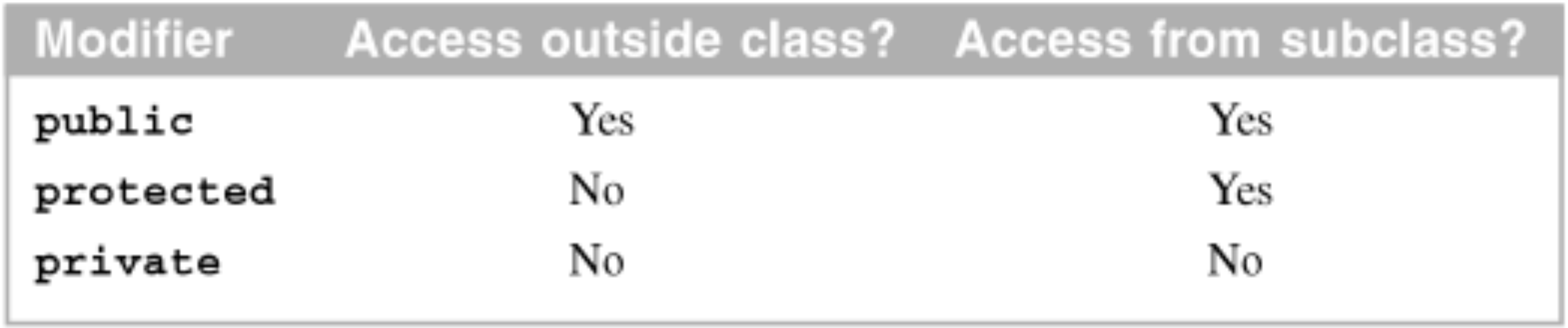
How to inherit a class (cont.)

- 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
```

How to use the protected access modifier

- Public and protected properties and methods are inherited by the subclass
- Private properties and methods are not inherited by the subclass.
- How the access modifiers work



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

How to use the protected access modifier (cont.)

- A superclass

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

    // The constructor and the get and set methods are the same
    // as the Person class in figure 14-13
}
```

- A subclass

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

    // The constructor and the get and set methods are the same
    // as the Employee class in figure 14-13

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

How to create abstract classes and methods

- An abstract class is a class that can’t be used to create an object.
- An abstract method is a method that specifies the name and parameters for the method but doesn’t provide a code block that implements the method.
- An abstract method can only be coded in an abstract class.
- A concrete class is a class that can be used to create an object.

How to create abstract classes and methods (cont.)

- An abstract class with an abstract method

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

    // The constructor and the get and set methods are the same
    // as the Person class in figure 14-13

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

How to create abstract classes and methods (cont.)

- A concrete class that implements 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; }

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

How to create abstract classes and methods (cont.)

- 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>';
```

How to create final classes and methods

- A final method cannot be overridden by a method in a subclass. As a result, all subclasses must use the final version of the method
- A final class cannot be inherited by a subclass
- How to prevent a method from being overridden
  - A class with a final method

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

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

How to create final classes and methods (cont.)

- A subclass that attempts to override a final method leading to a fatal error

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

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

How to create final classes and methods (cont.)

- How to prevent a class from being inherited
  - A final class

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

  - A class that attempts to inherit a final class leading to a fatal error

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

How to work with interfaces

- An interface defines a set of public methods that can be implemented by a class.
- All methods in an interface must be public and cannot be static.
- A class that implements an interface must provide an implementation for each method define by the interface
- An interface can define class constants that are available to any class that implements the interface.

How to work with interfaces (cont.)

- How to create an interface
  - The syntax

```
interface interfaceName {
    const contantName = contantValue;
    public function methodName( parameterList );
}
```

  - An interface to show an object

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

  - An interface to require two test methods

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

How to work with interfaces (cont.)

– An interface that provides two constants

```
interface Gender {
    const MALE = 'm';
    const FEMALE = 'f';
}
```

– A class that inherits a class and implements an

```
class Employee extends Person implements Showable {
    // The constructor and the get and set methods are the same
    // as the Person class in figure 14-13

    // 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, Gender { ... }
```

Summary

- In object-oriented programming, a class defines the properties and methods of each type of object.
- A constructor method, or just constructor, is a special method within a class that is used to create an object from the class.
- A destructor method is a special method that's executed when an object is no longer available for use.

Summary (2)

- An object is an instance of a class.
- A public property can be directly accessed by code outside of the class
- A private and protected properties can't be directly accessed by code outside the class
- A method is a function that's coded within a class.
- The object access operator (->) provides access to an object's properties and method.

Summary (3)

- A class constant is a constant value that belongs to the class, not to objects that are created from the class.
- A static property or static method is a property or method that belongs to a class, not to object created from the class.
- To create a second reference to an object, you can use the equals (=) operator.
- Create a copy of an object, PHP makes a shallow copy of the object

Summary (4)

- Inheritance provides a way to create a new class based on an existing class
- A class that inherits from a class is called a subclass, derived class, or child class.
- A subclass can extend the superclass by adding new properties and methods.
- An abstract class is a class that can’t be used to create an object.
- A concrete class is a class that can be used to create a object.

Summary (5)

- A final method can’t be overridden by a method in a subclass. A final class can’t be inherited by a subclass
- An interface defines a set of public methods that can be implemented by a class.