**Exercise 02\_10\_01 – Step 1**

In this Exercise, we will study object-oriented programming concepts. We will use objects in PHP scripts



1. Create a folder named Exercise 02\_10\_01 and open it with your IDE. Download the Student files from Google Classroom, unzip them, and place them in the project folder. Open up a clean server console on your LAMPP stack. Login to MySQL as root to open up the command line for MySQL Monitor as follows:  
   ***# mysql -h localhost -u root –p***
2. At the MySQL Monitor prompt, enter the following command to create a database and verify it was created:  
   ***> CREATE DATABASE onlinestores;  
   > USE OnlineStores;***  
   ***> SHOW DATABASES;***
3. At the MySQL Monitor prompt, enter the following command to create the ***storeinfo*** table:  
   ***> CREATE TABLE storeinfo   
    -> (storeID VARCHAR(10) PRIMARY KEY,   
    -> name VARCHAR(50), description VARCHAR(200),   
    -> welcome TEXT, cssFile VARCHAR(250),   
    -> email VARCHAR(100));***
4. At the MySQL Monitor prompt, enter the following command to view the structure of the new table:  
   ***> DESCRIBE storeinfo;***
5. Take the files ***StoreInfo.txt*** and ***Inventory.txt*** from the project folder and place them into the server ***mysql/data/onlinestores*** folder. At the MySQL Monitor prompt, enter the following command to load the data into the ***storeinfo*** table:  
   ***> LOAD DATA INFILE ‘StoreInfo.txt’   
    INTO TABLE storeinfo;***Go to phpMyAdmin and check the data.
6. At the MySQL Monitor prompt, enter the following command to create the ***storeinfo*** table:  
   ***> CREATE TABLE inventory   
    -> (storeID VARCHAR(10),   
    -> productID VARCHAR(10) PRIMARY KEY,   
    -> name VARCHAR(100), description VARCHAR(200),  
    -> price FLOAT);***
7. At the MySQL Monitor prompt, enter the following command to view the structure of the new table:  
   ***> DESCRIBE inventory;***
8. At the MySQL Monitor prompt, enter the following command to load the data into the ***inventory*** table:  
   ***> LOAD DATA INFILE ‘Inventory.txt’   
    INTO TABLE inventory;***Go to phpMyAdmin and check the data.

**Exercise 02\_10\_01 – Step 2**



1. Let’s begin to build a store page. Create a new file called ***GourmetCoffee.php***. Scaffold a basic HTML code layout into it. Complete our standard opening documentation in the ***<head>*** element. Make sure to have the ***modernizr*** <script> linked in. Set the <title> content to ***Gourmet Coffee***:  
   ***<!doctype html>  
   <html>  
   <head>  
    <title>Gourmet Coffee</title>  
    <meta charset="UTF-8">  
    <meta name="viewport" content="initial-scale=1.0">  
    <script src="modernizr.custom.65897.js"></script>  
   </head>  
   <body>  
   </body>  
   </html>***
2. Add the following text and elements to the document body:  
   <body>  
    ***<h1>Gourmet Coffee</h1>  
    <h2>Description goes here</h2>  
    <p>Welcome message goes here</p>  
    <p>Inventory goes here</p>***</body>  
   Give this a browser/server test.

**Exercise 02\_10\_01 – Step 3**



1. Let’s create an include file that will create a DB connection in object-oriented fashion. Create a new file called ***inc\_OnlineStoreDB.php***. Add a PHP script which will provide an array to store error messages and attempt the connection:  
   ***<?php  
   $errorMsgs = array();  
   $hostname = "localhost";  
   $username = "root";  
   $passwd = "";  
   $DBName = "onlinestores";  
   $DBConnect = new mysqli($hostname, $username, $passwd,   
    $DBName);?>***
2. Now let’s add the object-oriented technique to trap the connect error:  
   ***if ($DBConnect->connect\_error) {  
    $errorMsgs[] = "Unable to connect to the database server." .   
    " Error code " . $DBConnect->connect\_errno .   
    ": " . $DBConnect->connect\_error ;  
   }***
3. Return to ***GourmetCoffee.php*** and bring in the ***include*** file with the following script. Place it above the <!DOCTYPE> tag:  
   ***<?php  
   require\_once("inc\_OnlineStoreDB.php");  
   ?>***  
   <!doctype html>
4. Add the following PHP script to the document immediately before the closing ***</body>*** tag:  
    ***<?php  
    if (count($errorMsgs)) {  
    foreach ($errorMsgs as $msg) {  
    echo "<p>" . $msg . "</p>\n";  
    }  
    }  
     
    ?>***</body>
5. If we establish a connection, we must close it when we are done with it. We will do this with object-oriented technique. Add the following script to the bottom of the document:  
   </html>  
   ***<?php  
   if (!$DBConnect->connect\_error) {  
    echo "<p>Closing database   
    <em>$DBName</em>.</p>\n";  
    $DBConnect->close();  
   }***Give this a browser/server test with both a bad host and a good one.
6. Return to ***inc\_OnlineStoreDB.php***. Add the PHP **@** ***error*** ***control*** operator to the ***new*** statement to suppress PHP generated errors on the statement and use only our own custom messages:  
   $DBConnect = ***@***new mysqli($hostname, $username, $passwd,   
    $DBName);  
   Give this a browser/server test with both a bad host and a good one.

**Exercise 02\_10\_01 – Step 4**



1. Return to ***GourmetCoffee.php*** and remove the place holder for inventory:  
    ***~~<p>Inventory goes here</p>~~***
2. Add an object-oriented SQL ***query*** to the top of the PHP script below the placeholders. For flow of control, we will use a count of the error messages:  
    ***<?php  
    $TableName = "inventory";  
    if (count($errorMsgs) == 0) {  
    $SQLstring = "SELECT \* FROM $TableName" .   
    " WHERE storeID='COFFEE'";  
    $QueryResult = $DBConnect->query($SQLstring);  
    }***
3. Now let’s trap the SQL syntax failure case with the following if statement:  
    $QueryResult = $DBConnect->query($SQLstring);  
    ***if (!$QueryResult) {  
    $errorMsgs[] = "<p>Unable to perform the   
    query.<br>" .   
    "Error code " . $DBConnect->errno . ": " .   
    $DBConnect->error . "</p>\n";  
    }***  
   Give this a browser/server test with purposely bad SQL syntax, then good syntax.
4. We can now build out the query success case:  
    ***else {  
    echo "<table width='100%'>\n";  
    echo "<tr>\n";  
    echo "<th>Product</th>\n";  
    echo "<th>Description</th>\n";  
    echo "<th>Price Each</th>\n";  
    echo "</tr>\n";  
    while (($row = $QueryResult->fetch\_assoc()) != NULL)   
    {  
    echo "<tr><td>" . htmlentities($row['name']) .   
    "</td>\n";  
    echo "<td>" . htmlentities($row['description']) .   
    "</td>\n";  
    printf("<td>$%.2f</td></tr>\n", $row['price']);  
    }  
    echo "</table>\n";  
    }  
    }***Give this a browser/server test.

**Exercise 02\_10\_01 – Step 5**



1. Let’s create an include file that will define a custom ***class***. Create a new file called ***class\_OnlineStore.php***. Add a PHP script which will define the class with no members:  
   ***<?php  
   class OnlineStore {  
      
   }  
   ?>***
2. Return to ***GourmetCoffee.php*** and bring in the ***include*** file with a second require\_once() statement above the <!DOCTYPE> tag:  
   <?php  
   require\_once("inc\_OnlineStoreDB.php");  
   ***require\_once("class\_OnlineStore.php");***?>  
   Give this a browser server test for syntax.
3. Add the following code below the new ***require*** statement to ***instantiate*** an object from the class if it exists or trap an error:  
   require\_once("class\_OnlineStore.php");  
   ***if (class\_exists("OnlineStore")) {  
    $Store = new OnlineStore();  
   }  
   else {  
    $errorMsgs[] = "The <em>OnlineStore</em> class is not   
    available!";  
    $Store = NULL;  
   }***
4. Add the following code to the start of the PHP script in the ***<body>*** to determine that an object was instantiated from the class:  
    ***if ($Store != NULL) {  
    echo "<p>Successfully instantiated an object of the   
    <em>" . get\_class($Store) . "</em> class.</p>\n";  
    }***  
    $TableName = "inventory";  
   Give this a browser server test. First misspell the class to test the error trap. Then correct it.

**Exercise 02\_10\_01 – Step 6**



1. Return to the ***class\_OnlineStore.php***. Add some code to the PHP script which will add data members to the ***OnlineStore*** class. We will hide these properties by making them ***private***:  
   <?php  
   class OnlineStore {  
    ***private $DBConnect = NULL;  
    private $storeID = "";  
    private $inventory = array();  
    private $shoppingCart = array();***}  
   ?>   
   Give this a browser server test for syntax.

**Exercise 02\_10\_01 – Step 7**



1. Return to ***GourmetCoffee.php***. We will add some code to the top PHP script which will start a ***session*** and ***unserialize*** an OnlineStore object, if it has been serialized and stored in a session variable. If it has not, it will ***instantiate*** a new object. Add some debug code to test:  
   <?php  
   ***session\_start();***require\_once("inc\_OnlineStoreDB.php");  
   require\_once("class\_OnlineStore.php");  
   if (class\_exists("OnlineStore")) {  
    ***if (isset($\_SESSION['currentStore'])) {  
    echo "Unserializing object.<br>";  
    $Store = unserialize($\_SESSION['currentStore']);  
    }  
    else {  
    echo "Instantiating new object.<br>";  
    $Store = new OnlineStore();  
    }***  
   }  
   else {  
    $errorMsgs[] = "The <em>OnlineStore</em> class is not   
    available!";  
    $Store = NULL;  
   }  
   ?>
2. To serialize the object and store it as ***state***, add the following code to store it in ***$\_SESSION*** at the bottom of the ***else*** clause in the bottom PHP script:  
    echo "</table>\n";  
    ***$\_SESSION['currentStore'] = serialize($Store);***Give this a browser/server test. Go in and out of the app and check to see if it is instantiating or Unserializing. Once the session variable remains persistent, you can ***Clear storage*** on the Developer Tools Application tab to force an instantiation.

**Exercise 02\_10\_01 – Step 8**



1. Return to the ***class\_OnlineStore.php***. Add some code to the PHP script which will add a ***constructor*** function to the ***OnlineStore*** class:  
   <?php  
   class OnlineStore {  
    private $DBConnect = NULL;  
    private $storeID = "";  
    private $inventory = array();  
    private $shoppingCart = array();  
     **function \_\_construct() {  
      
    }**}  
   ?>
2. Let’s build out the ***constructor*** function. First we will bring in the external file that initializes the DB connection with an ***include***. A ***require()*** throws an error if not found, ***include()*** issues a warning. Then we employ an assignment statement using the ***$this*** reference to set the external DB connection object into an object property:  
    function \_\_construct() {  
    ***include("inc\_OnlineStoreDB.php");  
    $this->DBConnect = $DBConnect;*** }  
   Give this a browser/server test. Go in a couple of times to make sure it is all working with instantiating and serializing. Clear the cookie when done.

**Exercise 02\_10\_01 – Step 9**



1. Return to the ***class\_OnlineStore.php***. Add some code to the PHP script which will add a ***destructor*** function to the bottom of the ***OnlineStore*** class:  
    **function \_\_destruct() {  
      
    }**
2. Let’s build out the ***destructor*** function to close the DB connection:  
    function \_\_destruct() {  
    ***if (!$this->DBConnect->connect\_error) {  
    echo "<p>Closing database " .   
    "<em>$DBName</em>.</p>\n";  
    $this->DBConnect->close();  
    }*** }  
   Give this a browser/server test. The code throws an error for undefined variable $DBName on close().
3. Add the following property to the object:  
    private $DBConnect = NULL;  
    ***private $DBName = "";***
4. Initialize the property in the ***constructor*** function:  
    $this->DBConnect = $DBConnect;  
    ***$this->DBName = $DBName;***
5. Initialize the property in the ***constructor*** function:  
    $this->DBConnect = $DBConnect;  
    ***$this->DBName = $DBName;***
6. Change the syntax in the ***destructor*** function:  
    echo "<p>Closing database " .   
    "<em>***$this->DBName***</em>.</p>\n";  
   Give this a browser/server test. We now appear to have 2 closes of the DB connection on ***instantiation***. Going back in and ***unserializing***, we have 2 close notifications, but the second one, in the class, fails.
7. Return to ***GourmetCoffee.php*** and comment out the ***close()*** code in the bottom script:  
   ***//***if (!$DBConnect->connect\_error) {  
   ***//*** echo "<p>Closing database <em>$DBName</em>.</p>\n";  
   ***//*** $DBConnect->close();  
   **//**}  
   Give this a browser/server test. We now appear to have 1 working close of the DB connection on ***instantiation***. Going back in and ***unserializing***, we have 1 close notification, but it fails.

**Exercise 02\_10\_01 – Step 10**



1. After serialization, when the script ends, the ***\_\_destruct()*** function has been called on the OnlineStore object. On returning to the script, the object may be unserialized without calling the ***\_\_construct(***) function. To fix this, return to ***class\_OnlineStore.php***. Add some code to the bottom of the which will implement a ***wakeup()*** function:  
    **function \_\_wakeup() {  
      
    }**
2. Let’s build out the ***wakeup*** function to add some constructor code to fic the problem of the lost DB connection:  
    function \_\_wakeup() {  
    ***include("inc\_OnlineStoreDB.php");  
    $this->DBConnect = $DBConnect;  
    $this->DBName = $DBName;*** }  
   Give this a browser/server test. Remove the cookie and give the app a test on instantiation and unserialization. It appears everything is now working.

**Exercise 02\_10\_01 – Step 11**



1. Let’s add a ***setter*** function to the OnlineStore class. The purpose of it will be to set the private ***$storeID*** property. It wall also populate the private ***$inventory*** array. Let’s add some debug code to test a call to it:  
    **public function setStoreID($storeID) {  
    echo "\$storeID: $storeID<br>";  
    }**
2. Return to ***GourmetCoffee.php*** and we will add a variable to hold the ***storeID***, directly below the last ***require\_once()***:  
   require\_once("class\_OnlineStore.php");  
   ***$storeID = "COFFEE";***
3. Now we will make a call to the setter method to test it, directly below the code which ***instantiates*** or ***unserializes*** the store object:  
    else {  
    echo "Instantiating new object.<br>";  
    $Store = new OnlineStore();  
    }  
    ***$Store->setStoreID($storeID);***  
   Give this a browser/server test. Remove the cookie and give the app a test on instantiation and unserialization.
4. Back in the class file, let’s build out the ***setter***. Remove the debug code, and if the ***storeID*** has changed, let’s set the new ID, and build a query to return the ***inventory*** records. We will set up a ***failure*** handler and trap and test for SQL syntax problems:  
    public function setStoreID($storeID) {  
    ***if ($this->storeID != $storeID) {  
    $this->storeID = $storeID;  
    $TableName = "inventory";  
    $SQLstring = "SELECT \* FROM $TableName" .   
    " WHERE storeID='" .   
    $this->storeID . "'";  
    $queryResult =   
    $this->DBConnect->query($SQLstring);  
    if (!$queryResult) {  
    echo "<p>Unable to execute the query, " .   
    "error code: " .   
    $this->DBConnect->errno .   
    ": " . $this->DBConnect->error .   
    "</p>\n";  
    $this->storeID = "";  
    }  
    }***  
   Give this a browser/server test. Purposely misspell a word in the SQL string and test, then correct it and test again.
5. Now we will build the ***success*** scenario and populate the ***inventory*** array and initialize the ***shoppingCart*** array. We will output the results for debug:  
    ***else {  
    $inventory = array();  
    $shoppingCart = array();  
    while (($row = $queryResult->fetch\_assoc())   
    != NULL) {  
    $this->inventory[$row['productID']]   
    = array();  
    $this->inventory[$row['productID']]  
    ['name'] = $row['name'];  
    $this->inventory[$row['productID']]  
    ['description'] = $row['description'];  
    $this->inventory[$row['productID']]  
    ['price'] = $row['price'];  
    $this->shoppingCart[$row['productID']] = 0;  
    }  
    echo "<pre>\n";  
    print\_r($this->inventory);  
    print\_r($this->shoppingCart);  
    echo "</pre>\n"  
    }***  
   Give this a browser/server test. When it is working remove the debug code.
6. We will add a ***getter*** function to retrieve store info:  
    ***public function getStoreInformation() {  
    $retval = false;  
    if ($this->storeID != "") {  
    $TableName = "storeinfo";  
    $SQLstring = "SELECT \* FROM $TableName" .   
    " WHERE storeID='" .   
    $this->storeID . "'";  
    $queryResult =   
    $this->DBConnect->query($SQLstring);  
    if ($queryResult) {  
    $retval = $queryResult->fetch\_assoc();  
    }  
    }  
    return $retval;  
    }***
7. Return to ***GourmetCoffee.php*** and we will add an array variable to hold the ***store information***, below the last ***require\_once()***:  
   require\_once("class\_OnlineStore.php");  
   $storeID = "COFFEE";  
   ***$storeInfo = array();***
8. Now we will make a call to the ***getter*** method to test it, directly below the call to the first setter method. Add some debug to see the results:  
    $Store->setStoreID($storeID);  
    ***$storeInfo = $Store->getStoreInformation();  
    echo "<pre>\n";  
    print\_r($storeInfo);  
    echo "</pre>\n";***  
   Give this a browser/server test. When it is working remove the debug code.
9. We will add another ***getter*** function that will retrieve data from the ***$inventory*** array property It will also format a table for the inventory product list. We will also add a new column that will hold a hyperlink to select an item for the shopping cart:  
    ***public function getProductList() {  
    $retval = false;  
    $subtotal= 0;  
    if (count($this->inventory) > 0) {  
    echo "<table width='100%'>\n";  
    echo "<tr>";  
    echo "<th>Product</th>";  
    echo "<th>Description</th>";  
    echo "<th>Price Each</th>";  
    echo "<th># in Cart</th>";  
    echo "<th>Total Price</th>";  
    echo "<th>&nbsp;</th>";  
    echo "</tr>\n";  
    foreach ($this->inventory as $ID => $info) {  
    echo "<tr>";  
    echo "<td>" . htmlentities($info['name']) .   
    "</td>";  
    echo "<td>" .   
    htmlentities($info['description']) .   
    "</td>";  
    printf("<td class='currency'>$%.2f</td>",   
    $info['price']);  
    echo "<td class='currency'>" .   
    $this->shoppingCart[$ID] . "</td>";  
    printf("<td class='currency'>$%.2f</td>",   
    $info['price'] \*   
    $this->shoppingCart[$ID]);  
    echo "<td><a href='" .   
    $\_SERVER['SCRIPT\_NAME'] .   
    "?PHPSESSID=" . session\_id() .  
    "&ItemToAdd=$ID’>Add Item</a></td>";  
    $subtotal += ($info['price'] \*   
    $this->shoppingCart[$ID]);  
    echo "</tr>\n";  
    }  
    echo "<tr><td colspan='4'>Subtotal</td>";  
    printf("<td class='currency'>$%.2f</td>",  
    $subtotal);  
    echo "<td>&nbsp;</td></tr>";  
    echo "</table>\n";  
    $retval = true;  
    }  
    return($retval);  
    }***  
   Give this a browser/server test for syntax errors.
10. We will return to ***GourmetCoffee.php*** to make some more use of the methods that we ***encapsulated*** into the object. We will begin by linking in a ***stylesheet*** below the ***<script>*** tag, using info from the database to pick the correct stylesheet:  
     <script src="modernizr.custom.65897.js"></script>  
     ***<link rel="stylesheet" type="text/css"   
     href="<?php echo $storeInfo['cssFile']; ?>">***Give this a browser/server test to make sure the stylesheet is working.
11. We will now modify the first three ***<body>*** tags to further use the data from the object:  
     ***<h1><?php echo htmlentities($storeInfo['name']);   
     ?></h1>  
     <h2><?php echo htmlentities($storeInfo['description']);   
     ?></h2>  
     <p><?php echo htmlentities($storeInfo['welcome']);   
     ?></p>***Give this a browser/server test to make sure the page is displaying correctly.
12. ***Comment*** out the entire contents of the PHP script below the three lines above. Replace the content with the following ***method*** call and assignment of the ***$\_SESSION*** variable:  
     <p><?php echo htmlentities($storeInfo['welcome']); ?></p>  
     <?php  
     ***$Store->getProductList();  
     $\_SESSION['currentStore'] = serialize($Store);***Give this a browser/server test to make sure the new table is displaying correctly. Test the hyperlink and make sure we are returned to the app and that the URL is correct.
13. Comment out the first ***require\_once()*** call, as the entire functionality of it is now ***encapsulated*** in the class:  
    session\_start();  
    ***//require\_once("inc\_OnlineStoreDB.php");***Give this a browser/server test.

**Exercise 02\_10\_01 – Step 12**



1. Let’s add a ***utility*** function to the end of the OnlineStore class in ***class\_OnlineStore.php***. The purpose of it will be to add an item to the shopping cart:  
    ***public function addItem() {  
    $prodID = $\_GET['ItemToAdd'];  
    if (array\_key\_exists($prodID, $this->shoppingCart)) {  
    $this->shoppingCart[$prodID] += 1;   
    }  
    }***
2. Return to ***GourmetCoffee.php*** and we will modify the code to add an item to the shopping cart when a user clicks the hyperlink:  
    $storeInfo = $Store->getStoreInformation();  
    ***if (isset($\_GET['ItemToAdd'])) {  
    $Store->addItem();  
    }***Give this a browser/server test. Make sure the ***# in Cart***, ***Total Price***, and ***Subtotal*** properly update.

**Exercise 02\_10\_01 – Step 13**



1. Let’s demonstrate code reuse with OOP. Duplicate ***GourmetCoffee.php*** to a new file named ***OldTypeAntiques.php***. Change the assignment of the $StoreID variable as follows:  
   require\_once("class\_OnlineStore.php");  
   ***$storeID = "ANTIQUE";***Give this a browser/server test.
2. Let’s demonstrate code reuse with OOP. Duplicate ***GourmetCoffee.php*** to a new file named ***ElectronicsBoutique.php***. Change the assignment of the $StoreID variable as follows:  
   require\_once("class\_OnlineStore.php");  
   ***$storeID = "ELECBOUT";***Give this a browser/server test.