PROJECT #2: "Store Inventory" v1.1

Topic: "Creating classes"

PART 1: "Create the Database class"

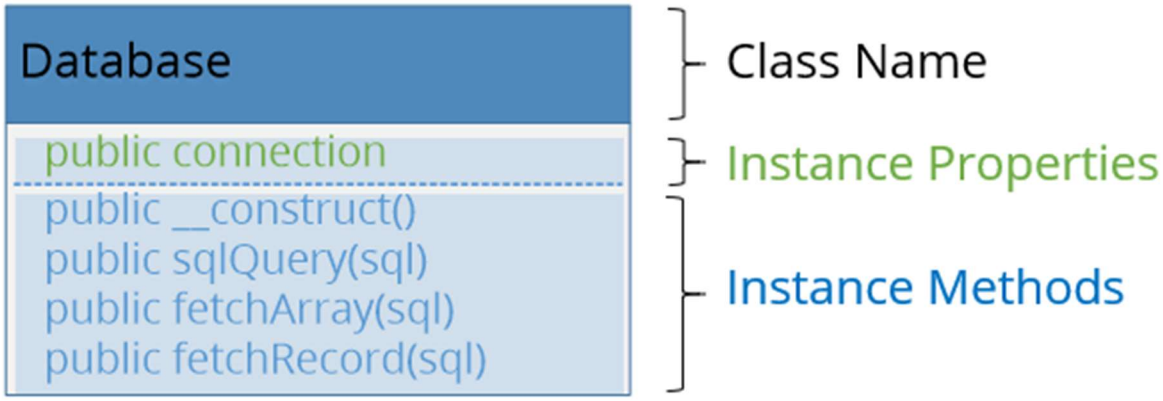
Objectives:

- Create a Database Class
- Update bootstrap.php file to include the Database Class

Instructions:

1. Create a new file called **Database.php** in the "**includes**" sub directory. Inside the file, define the skeleton for a new class called **Database**.

   Recall that a class is made up of instance variables also known as properties (state) and instance methods (behaviour).



In the **Database** class definition, you will create the following instance properties and instance methods based on the UML diagram below:
The database class contains only one instance property the open connection to the database.

The database class contains the following methods:

- A constructor method
- A method called sqlQuery which has one parameter. This method will be used to **Create**, **Update** and **Delete** records from the database.
- A method called fetchArray which has one parameter. This method will be used to **Retrieve** a list of records from the database.
- A method called fetchRecord which has one parameter. This method will be used to **Retrieve** a single record from the database.

Defining Database Class Properties

2. Within the **Database** class definition, create the public instance property as illustrated in the UML diagram.

Defining Database Class Methods

Method: __construct()

Within the **Database** class definition, create a public **constructor** method. The purpose of this method is to create a new connection to the database and store it into a property when we instantiate an object of the **Database** class.

3. Within the constructor method, use the PDO method to create a new connection to the database. Follow the same syntax used in **createMovieTable.php**.

4. Store this new connection object in the property **$connection** which you created earlier.

```php
/*
 * Database Class
 */

class Database {

    public $connection;

    /**
     * Database constructor.
     * Creates a connection to the database using
     * PDO methods and stores it in an instance variable
     */
    public function __construct() {
        // Create a new connection using PDO methods and store it
        // into the property $connection
    }
```

Method: sqlQuery

Create a public method called **sqlQuery**. This methods purpose will be to execute SQL statements passed to it as a parameter. **Note:** this will only be used for Create, Update or Delete operations.

The method has **one** parameter; a variable named **$sql**. This is SQL statement that is to be executed.

5. Within the body of the method, use the connection to the database (stored in the instance variable) to query the SQL statement stored in **$sql**. Store the result of the query into variable called **$result**.

6. Return the variable **$result**. This is a PDO object that contains the results of the SQL statement that was executed on the database.

```php
/**
 * Executes the provided SQL statement
 * @param $sql
 * @return bool|PDOStatement
 */
public function sqlQuery($sql) {
    $dbc = $this->connection;
    $result = $dbc->query($sql);
    return $result;
}
```

Method: fetchArray

Create a public method called **fetchArray**. This methods purpose will be to retrieve results from an SQL query. The results will be placed in an array. **Note:** this will only be used for retrieving **multiple** records from a table.

The method has **one** parameter; a variable named **$sql**. This is SQL statement that is to be executed.

7. Within the body of the **fetchArray** method, call the **sqlQuery** method to execute the SQL query. Store the results in a variable called **$result**.

8. Invoke the PDO method **rowCount()** on the **$result** variable to determine the number of rows that were returned by the database query.

9. Use a conditional statement to test the number of rows:

    a. If there are 0 rows returned, return false.
    b. Otherwise, use the PDO method **fetchAll(PDO::FETCH_ASSOC)** to return an associative array of all the records.

```php
/**
 * Executes SQL statements and returns results into array
 * of assoc. arrays
 * @param $sql
 * @return array|bool
 */
public function fetchArray($sql) {
    $result = $this->sqlQuery($sql);
    $numberOfRows = $result->rowCount();
    if ($numberOfRows == 0) {
        return false;
    } else {
        $resultArray = $result->fetchAll(PDO::FETCH_ASSOC);
        return $resultArray;
    }
}
```

Method: fetchRecord

Create a public method called **fetchRecord**. This method is very similar to fetchArray, however, its purpose is to retrieve a single record.

The method has **one** parameter; a variable named **$sql**. This is an SQL statement that is to be executed.

10. Within the body of the method, call the **sqlQuery** method to execute the SQL query. Store the resulting object in a variable called **$result**.

11. Use the PDO method **rowCount()** on the **$result** variable to determine the number of rows that were returned by the query.

12. Use a conditional statement to test the number of rows:

    a. If there are 0 rows returned, return false.
    b. Otherwise, use the PDO method **fetch(PDO::FETCH_ASSOC)** to return single record in an associative array.

```php
/**
 * Execute the provided SQL statement and retrieve results into assoc.
 * array
 * @param $sql
 * @return bool|mixed
 */
public function fetchRecord($sql) {
    $result = $this->sqlQuery($sql);
    $numberOfRows = $result->rowCount();
    if ($numberOfRows == 0) {
        return false;
    } else {
        $resultRow = $result->fetch(PDO::FETCH_ASSOC);
        return $resultRow;
    }
}
```

All the properties and methods that we need for the **Database** class are now complete.

13. Outside of the class definition, instantiate a new object of class **Database** called **$dbc**. We can now use this active database connection throughout the rest of our project.

```php
} //End of Database Class

$dbc = new Database();

?>
```

14. Post the **Database.php** file to the server in the **/public_html/inventory_oop/includes/** directory.

15. Open the **bootstrap.php** file. Add the **Database.php** to the list of required files needed to initialize the inventory.

Note: If you forget to include this file, PHP will produce a fatal error.

You’re now ready to move on to the next section.