

Prepared by Mr. Renato L. Adriano II

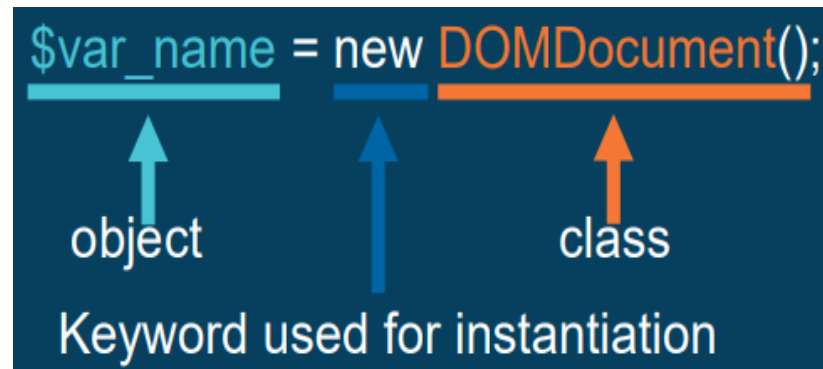
The background of the slide is a watercolor-style splash. It features a large, irregular shape in the center, filled with a gradient from bright orange at the top to a deep blue at the bottom. This central shape is surrounded by a lighter, textured wash of the same colors, with small droplets and splatters extending outwards onto the white background.

XML DOM USING PHP

The DOMDocument() class

The **DOMDocument()** class is a class already available in PHP that allows the manipulation of a markup language, specifically, an XML file.

SYNTAX:



```
$var_name = new DOMDocument();
```

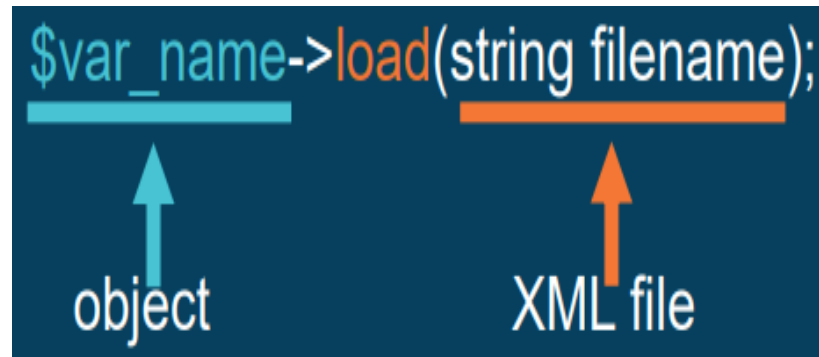
The diagram shows the syntax `$var_name = new DOMDocument();` with three underlined components: `$var_name` (light blue), `new` (dark blue), and `DOMDocument()` (orange). Below these, three arrows point upwards: a light blue arrow from the word 'object' to `$var_name`, a dark blue arrow from the text 'Keyword used for instantiation' to `new`, and an orange arrow from the word 'class' to `DOMDocument()`.

NOTE: Creates a new DOM Document to hold XML structure.

The load() METHOD

The **load()** method is used to identify which file shall be loaded and be parsed by the DOMDocument() class. The file name or the file's URL shall be provided as its argument.

SYNTAX:



```
$var_name->load(string filename);
```

The diagram illustrates the syntax of the `load()` method. It shows the code `$var_name->load(string filename);` on a dark blue background. A light blue horizontal line underlines `$var_name`, and a light orange horizontal line underlines `load`. A light blue arrow points from the word `object` below to `$var_name`. A light orange arrow points from the text `XML file` below to `load`.

NOTE: `load()` method accepts file locations. If the PHP file and XML file are not in the same folder, provide the path or URL to the string argument.



APPLYING CRUD IN PHP XML DOM

```

<body>
  <xml id="xmldata" style="display:none;">
    <students>
      <student id="20180602">
        <name>John Dela Cruz</name>
        <birthday>January 1, 1998</birthday>
        <course>BSIT</course>
      </student>
      <student id="20190818">
        <name>Jane Santos</name>
        <birthday>June 12, 1999</birthday>
        <course>BSIT</course>
      </student>
      <student id="20170408">
        <name>Ryan Reyes</name>
        <birthday>May 8, 1997</birthday>
        <course>BSIT</course>
      </student>
      <student id="20160701">
        <name>Carlo Ople</name>
        <birthday>July 1, 1996</birthday>
        <course>BSIT</course>
      </student>
    </students>
  </xml>
</body>

```

- This is the file to be parsed by the PHP XML DOM Parser, the DOMDocument() Class.

- For this example, this file is saved as students.xml.

XML FILE: students.xml

The background features a large, abstract watercolor splash. The top half is a vibrant orange, which transitions into a deep blue at the bottom. The edges are irregular and textured, with smaller splatters of the same colors scattered across the white background.

CREATE
(DATA CREATION)

```
<!doctype html>
<html>
  <head>
    <title>Create Page</title>
  </head>
  <body>
    <form method="post" action="createProcess.php">
      Student No.: <input type="text" name="studId"/><br/>
      Full Name: <input type="text" name="fullName"/><br/>
      Birthday: <input type="text" name="birthday"/><br/>
      Course: <input type="text" name="course"/><br/>
      <br/><input type="submit" value="Save">
    </form>
  </body>
</html>
```

STEP 1: CREATE create.php

First, create the form that will accept information from the user.

Student No.: 20200416

Full Name: Oliver Austria

Birthday: April 16, 2000

Course: BSIT

Save

OUTPUT

<?php

```
$xml = new domdocument("1.0");  
$xml->load("students.xml");  
  
$studId = $_POST["studId"];  
$fullName = $_POST["fullName"];  
$birthday = $_POST["birthday"];  
$course = $_POST["course"];  
  
$student = $xml->createElement("student");  
$name = $xml->createElement("name", $fullName);  
$bday = $xml->createElement("birthday", $birthday);  
$crs = $xml->createElement("course", $course);  
  
$student->setAttribute("id", $studId);  
$student->appendChild($name);  
$student->appendChild($bday);  
$student->appendChild($crs);  
  
$xml->getElementsByTagName("students")->item(0)->appendChild($student);  
$xml->save("students.xml");  
echo "Record saved...<a href='create.php'>Back</a>";
```

- After creating the form, create the file that will process the user input. For this example, the file will be saved as createProcess.php.

STEP 2:

CREATE createProcess.php

OUTPUT

Student No.: 20180602
Full Name: John Dela Cruz
Birthday: January 1, 1998
Course: BSIT

Student No.: 20190818
Full Name: Jane Santos
Birthday: June 12, 1999
Course: BSIT

Student No.: 20170408
Full Name: Ryan Reyes
Birthday: May 8, 1997
Course: BSIT

Student No.: 20160701
Full Name: Carlo Ople
Birthday: July 1, 1996
Course: BSIT

Student No.: 20200416
Full Name: Oliver Austria
Birthday: April 16, 2000
Course: BSIT



CODE EXPLANATION

(createProcess.php)

```
$xml = new domdocument();  
$xml->load("students.xml");
```

createProcess.php

- Sets up the PHP XML DOM Parser. Instantiate the class, then load the XML file.

```
$studId = $_POST["studId"];  
$fullName = $_POST["fullName"];  
$birthday = $_POST["birthday"];  
$course = $_POST["course"];
```

createProcess.php

- Gets the user input using the super global variable \$_POST.

```
$student = $xml->createElement("student");  
$name = $xml->createElement("name",$fullName);  
$bday = $xml->createElement("birthday",$birthday);  
$crs = $xml->createElement("course",$course);
```

createProcess.php

- Create the elements using the createElement() method. No element node will be created for the ID as the ID will be an attribute.
- createElement() method creates a new element node.

SYNTAX: createElement(node_name , node_value)

```
$student->setAttribute("id",$studId);
```

createProcess.php

- Create the attribute using the setAttribute() method.
- setAttribute() method sets the value to the specified attribute.

SYNTAX: setAttribute("attr_name", attr_value)


```
$student->appendChild($name);  
$student->appendChild($bday);  
$student->appendChild($crs);
```

createProcess.php

- After the elements were created, include the created elements on the <student> element node using the appendChild() method.
- appendChild() method adds an element at the end of the element (will be the last child of the element).

SYNTAX: appendChild(node_name)

```
$xml->getElementsByTagName("students")->item(0)->appendChild($student);  
$xml->save("students.xml");  
echo "Saved!";
```

createProcess.php

- Last few things to do is to add the newly created <student> element to the root element (<students>), then save the file using the save() method.
- save() method accepts the location of the file to be saved.

SYNTAX: save(string url)



READ
(DATA RETRIEVAL)

```
<?php
$xml = new domdocument("1.0");
$xml->load("students.xml");

$students = $xml->getElementsByTagName("student");

foreach($students as $student)
{
    $id = $student->getAttribute("id");
    $name = $student->getElementsByTagName("name")->item(0)->nodeValue;
    $birthday = $student->getElementsByTagName("birthday")->item(0)->nodeValue;
    $course = $student->getElementsByTagName("course")->item(0)->nodeValue;

    echo "<b>Student No.: </b> $id<br>";
    echo "<b>Full Name: </b> $name<br>";
    echo "<b>Birthday: </b> $birthday<br>";
    echo "<b>Course: </b> $course<br>";
    echo "<br>";
}
?>
```

CREATE read.php

OUTPUT

Student No.: 20180602
Full Name: John Dela Cruz
Birthday: January 1, 1998
Course: BSIT

Student No.: 20190818
Full Name: Jane Santos
Birthday: June 12, 1999
Course: BSIT

Student No.: 20170408
Full Name: Ryan Reyes
Birthday: May 8, 1997
Course: BSIT

Student No.: 20160701
Full Name: Carlo Ople
Birthday: July 1, 1996
Course: BSIT

Student No.: 20200416
Full Name: Oliver Austria
Birthday: April 16, 2000
Course: BSIT

The background features a large, abstract watercolor splash. The top portion is a vibrant orange, which transitions into a deep blue at the bottom. The edges of the splash are irregular and textured, with smaller droplets and splatters extending outwards. The text is centered within the orange section.

CODE EXPLANATION

(read.php)

```
$xml = new domdocument();  
$xml->load("students.xml");
```

read.php

- First thing to do is to set up the PHP XML DOM Parser. Instantiate the class, then load the XML file.

```
$students = $xml->getElementsByTagName("student");
```

read.php

- Get the <student> elements on the XML file. After getting all the <student> elements, which are stored in \$students variable, loop through it using foreach.

NOTE: \$students variable is an array, it contains all <student> elements.


```
foreach ($students as $student) {  
    $id = $student->getAttribute("id");  
    $name = $student->getElementsByTagName("name")->item(0)->nodeValue;  
    $birthday = $student->getElementsByTagName("birthday")->item(0)->nodeValue;  
    $course = $student->getElementsByTagName("course")->item(0)->nodeValue;  
  
    echo "<b>Student No.:</b> $id<br>";  
    echo "<b>Full Name:</b> $name<br>";  
    echo "<b>Birthday:</b> $birthday<br>";  
    echo "<b>Course:</b> $course<br>";  
    echo "<br>";  
}
```

read.php

- `getElementsByTagName()` method gets all elements with the specified tag name.
- `getAttribute()` method allows the retrieval of the stored value in the specified attribute.
- `item()` method allows the selection of a specific value using index to an array.
- `nodeValue` property returns the stored value in an element.

READ
(DATA
SEARCH)



```
<!doctype html>
<html>
  <head>
    <title>Search Page</title>
  </head>
  <body>
    <form method="post" action="searchProcess.php">
      Student No.: <input type="text" name="studId"/><br/>
      <br/><input type="submit" value="Search">
    </form>
  </body>
</html>
```

STEP 1: CREATE search.php

First, create the form that will accept information from the user.

A

Student No.: 20170408

Search

B

Student No.: 123456789

Search

OUTPUT

```

<?php
$xml = new domdocument("1.0");
$xml->load("students.xml");
$students = $xml->getElementsByTagName("student");

$flag = 0;
$search = $_POST["studId"];

foreach($students as $student)
{
    $id = $student->getAttribute("id");
    if($search == $id)
    {
        $flag = 1;

        $id = $student->getAttribute("id");
        $name = $student->getElementsByTagName("name")->item(0)->nodeValue;
        $birthday = $student->getElementsByTagName("birthday")->item(0)->nodeValue;
        $course = $student->getElementsByTagName("course")->item(0)->nodeValue;

        echo "<b>Student No.: </b> $id<br>";
        echo "<b>Full Name: </b> $name<br>";
        echo "<b>Birthday: </b> $birthday<br>";
        echo "<b>Course: </b> $course<br>";
        echo "<a href='search.php'>Back</a>";
        break;
    }
}

if($flag == 0) echo "No record found.<a href='search.php'>Back</a>";
?>

```

STEP 2: CREATE searchProcess.php



Student No.: 20170408
Full Name: Ryan Reyes
Birthday: May 8, 1997
Course: BSIT



No record found. [Back](#)

OUTPUT



CODE EXPLANATION

(searchProcess.php)

```
$flag = 0;  
$search = $_POST["studId"];
```

searchProcess.php

- \$flag will determine if a matching record is found or not.
- \$search will store the value from the textbox in search.php


```
$id = $student->getAttribute("id");  
if($id == $search)  
{  
    $flag = 1;  
}
```

searchProcess.php

- \$id stores the value of the student number for the current student element.
- The if statement will test if the current student number matches the student number the user is searching.
- \$flag is set to 1 to indicate that a matching record has been found in the XML file.

```
        break;
    }
}
if($flag == 0) echo "No record found.<a href='search.php'>Back</a>";
```

searchProcess.php

- break statement is used to leave the loop and no longer finish its iteration.
- The if statement will test if the value of \$flag remains at 0, this means that no matching record has been found.

The background features a large, abstract watercolor splash. The top half is a vibrant orange, which transitions into a deep blue at the bottom. The edges are irregular and textured, with smaller splatters of the same colors scattered across the white background.

UPDATE
(DATA MODIFICATION)

```

<!doctype html>
<html>
  <head>
    <title>Update Page</title>
  </head>
  <body>
    <form method="post" action="updateProcess.php">
      Student No.: <select name="id">
        <option>Select ID</option>
        <?php
          $xml = new domdocument("1.0");
          $xml->load("students.xml");
          $students = $xml->getElementsByTagName("student");


          foreach($students as $student){
            $id = $student->getAttribute("id");
            echo "<option>" . $id. "</option>";
          }
        ?>
      </select><br>
      New Birthday: <input type="date" name="newBirthday"/><br/>
      New Course: <input type="text" name="newCourse"/><br/>
      <br/><input type="submit" value="Update">
    </form>
  </body>
</html>

```

STEP 1: CREATE update.php

First, create the form that will accept information from the user.

Student No.: 20200416 ▾

New Birthday: 25 Dec 2000 

New Course: ARCHITECTURE

Update

OUTPUT

The background features a large, irregular watercolor splash. The top portion is a vibrant orange, which transitions into a deep blue at the bottom. The edges of the splash are textured and feathered, with small droplets and splatters extending outwards. The overall effect is artistic and dynamic.

CODE EXPLANATION

(update.php)

```
<?php
$xml = new domdocument ("1.0");
$xml->load ("students.xml");
$students = $xml->getElementsByTagName ("student");

foreach ($students as $student) {
    $id = $student->getAttribute ("id");
    echo "<option>" . $id. "</option>";
}
?>
```

update.php

- This PHP code serves as the retrieval of IDs of the students in the records. This was done to avoid typing the student number to lessen input errors by the users.

```
1 <?php
2 $xml = new domdocument("1.0");
3 $xml->load("students.xml");
4 $students = $xml->getElementsByTagName("student");
5
6 $id = $_POST["id"];
7 $birthday = $_POST["newBirthday"];
8 $course = $_POST["newCourse"];
9 $flag = 0;
10
11 foreach($students as $student)
12 {
13     $oldid = $student->getAttribute("id");
14
15     if( $id == $oldid)
16     {
17         $flag = 1;
18         $name = $student->getElementsByTagName("name")->item(0)->nodeValue;
```

STEP 2: |
CREATE updateProcess.php |


```
20 $newNode = $xml->createElement("student");
21 $newNode->setAttribute("id", "$id");
22 $newNode->appendChild($xml->createElement("name", $name));
23 $newNode->appendChild($xml->createElement("birthday", $birthday));
24 $newNode->appendChild($xml->createElement("course", $course));
25
26 $oldNode = $student;
27
28 $xml->getElementsByTagName("students")->item(0)->replaceChild($newNode, $oldNode);
29 echo "Student ID: $id<br>Student's Name: $name<br><br>";
30 echo "Details Updated...<br><a href='update.php'>Back</a>";
31 $xml->save("students.xml");
32 break;
33 }
34 }
35 if($flag == 0) echo "Modification failed...<a href='update.php'>Back</a>";
36 ?>
```

STEP 2: Continuation

Student ID: 20200416

Student's Name: Oliver Austria

Details Updated...

[Back](#)

OUTPUT



CODE EXPLANATION

(updateProcess.php)

```
$flag=1;
```

```
$name = $student->getElementsByTagName("name")->item(0)->nodeValue;
```

updateProcess.php

- Inside the if, since we know that the modification will be done here, we can set the flag to 1.
- Gets the name of the student since it is not part of the data to be modified.

```
$newNode = $xml->createElement('student');  
$newNode->setAttribute('id',$id);  
$newNode->appendChild($xml->createElement('name',$name));  
$newNode->appendChild($xml->createElement('birthday',$birthday));  
$newNode->appendChild($xml->createElement('course',$course));
```

updateProcess.php

- Creates the new node for the updated records of the student.

```
$oldNode = $student;
```

updateProcess.php

- Determines the old node of the student.

NOTE: Determining the old and new nodes are required for the method that we will use in modification.

```
$xml->getElementsByName('students')->item(0)->replaceChild($newNode,$oldNode);  
$xml->save("students.xml");  
break;
```

updateProcess.php

- replaceChild() method accepts nodes as source and destination.

SYNTAX: replaceChild(newNode, oldNode)

- break statement is used to leave the loop and no longer finish its iteration.

REMEMBER



The **new node** will be the replacement to the **old node**. With this action, it acts as if the old node was modified, but in reality, it was replaced with a new node containing the updated values!

The background features a large, abstract watercolor splash. The top half is a vibrant orange, which transitions into a deep blue at the bottom. The edges are irregular and splattered, with smaller droplets of color scattered across the white background.

DELETE
(DATA DELETION)

```

<!doctype html>
<html>
  <head>
    <title>Delete Page</title>
    <script>
      function deletion() {
        var choice = confirm("Do you really want to delete this record?");
        if(choice==false)
          return false;
      }
    </script>
  </head>
  <body>
    <form method="post" action="deleteProcess.php" onsubmit="return deletion()">
      Student No.: <select name="id">
        <option>Select ID</option>
        <?php
          $xml = new domdocument("1.0");
          $xml->load("students.xml");
          $students = $xml->getElementsByTagName("student");

          foreach($students as $student){
            $id = $student->getAttribute("id");
            echo "<option>" . $id. "</option>";
          }
        ?>
      </select><br>
      <br/><input type="submit" value="Delete">
    </form>
  </body>
</html>

```

STEP 1: CREATE delete.php

First, create the form that will accept information from the user.

Student No.: 20200416 ▾

Delete

localhost says

Do you really want to delete this record?

OK

Cancel

OUTPUT

The background features a large, abstract watercolor splash. The top half is a vibrant orange, which transitions into a deep blue at the bottom. The edges are irregular and splattered, with smaller droplets of color scattered across the white background.

CODE EXPLANATION

(delete.php)

```
<script>
    function deletion() {
        var choice = confirm("Do you really want to delete this record?");
        if(choice==false)
            return false;
    }
</script>
```

```
<form method="post" action="deleteProcess.php" onsubmit="return deletion()">
```

delete.php

- function deletion() generates a confirmation window that will ask if you really want to delete a record.
- If “return false” is detected during form submission, it will not continue with the submission.
- onsubmit=“return deletion” triggers the function “deletion()” during form submission.

```
<?php
$xml = new domdocument("1.0");
$xml->load("students.xml");
$students = $xml->getElementsByTagName("student");

$id = $_POST["id"];

foreach($students as $student)
{
    $oldid = $student->getAttribute("id");

    if($id == $oldid)
    {
        $xml->getElementsByTagName("students")->item(0)->removeChild($student);
        $xml->save("students.xml");
        echo "Record deleted!</br><a href='delete.php'>Back</a>";
        break;
    }
}

?>
```

STEP 2: |
CREATE deleteProcess.php |

Record deleted!

[Back](#)

OUTPUT

The background of the slide is a large, abstract watercolor splash. It features a mix of orange, red, and blue hues, with the colors blending into each other. The splash has a textured, painterly appearance with visible brushstrokes and splatters. The text is centered within this splash.

CODE EXPLANATION

(deleteProcess.php)


```
$xml->getElementsByTagName('students')->item(0)->removeChild($student);
```

deleteProcess.php

- The removeChild() method is used to remove the element node of the selected ID.
- The removeChild() method removes a child node from its parent node.

SYNTAX: removeChild(nodeName)



END OF LESSON