|  |  |
| --- | --- |
|  | Department of Information Technology, Malang State Polytechnic  **Jobsheet-12: PHP – OOP**  **Web Design and Programming Courses**  Tutor: Web Design and Programming Teaching Team  *November 2024* |

**Dimas Arya Sadewa**

**2341760173 – 08**

**SIB 2G**

**Topic**

* Dynamic Web Development Concepts with OOP

**Purpose**

Students are expected to:

1. Students are able to create classes and objects, inheritance, polymorphism, encapsulation, abstraction, interfaces, constructors and destructors, and encapsulation and access modifiers
2. Students are able to create CRUD with OOP

**Attention**

This jobsheet must be done step-by-step according to the practicum steps that have been given. Questions can be answered directly in the column provided using the PDF Editor.

**Introduction**

**OOP**

Object-Oriented Programming (OOP) is a very important programming paradigm in the world of software development. This allows developers to organize their code into objects that have associated attributes (data) and methods (functions).

**Introduction to Object-Oriented Programming (OOP)**

Object-Oriented Programming is based on the concept of objects, which represent entities in the real world. Each object has characteristics called attributes (properties), and can perform certain actions called methods (functions). OOP helps in breaking down the code into smaller, more manageable parts.

## Why is OOP Important?

In the increasingly complex and dynamic world of website development, the use of Object-Oriented Programming (OOP) Concepts has become an essential foundation. OOP brings invaluable effectiveness, ease of maintenance, and scalability to website projects. This article will discuss why OOP is so important in website project development and its key benefits.

### Modularity and Better Code Management

One of the main benefits of OOP is its ability to break code into independent modules or objects. In website development, each component such as forms, views, databases, and more can be represented as separate objects. This allows the development team to work separately on these components, speeding up the development process and allowing for easier maintenance in the future.

### Reusability and Efficiency

In OOP, objects can be reused in different parts of the project. This reduces the amount of code that needs to be written, saving developers time and effort. For example, if you've created a "Form" object that has a method for validating input, you can use it on various pages of your website without needing to rewrite that validation code.

### Better Error Management

When an error occurs in OOP code, you can easily isolate and find the source of the error because each object has a clear responsibility. This allows you to fix problems faster and more accurately, reducing the time spent on debugging.

### Scalability and Collaborative Development

Website projects tend to evolve over time. With OOP, you can easily add new features or update existing components without interfering with other functions. The development team can also work in parallel on various components, as each object stands alone and is less dependent on the other.

### Easier Maintenance

As a website project grows, maintenance becomes very important. OOP helps in separating the necessary changes to one component without affecting the others. If you want to change the appearance of a particular page, you just need to edit the view object without worrying about the impact on other components.

### Encapsulation and Security

The concept of encapsulation in OOP allows you to hide implementation details from other components. This means that other components can only interact with objects through defined interfaces, reducing the potential for errors or unwanted manipulation.

### Flexibility and Improved Code Quality

OOP allows you to create high abstractions to manage complexity and define common patterns. It improves code quality because it follows proven principles in software design, such as DRY (Don't Repeat Yourself) and SOLID (Split-Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, Version Dependency).

## Key Concepts in OOP PHP

In PHP, OOP allows you to organize and group code into more structured, manageable units. Here are the main concepts of OOP in PHP:

**Practicum 1. Basic OOP**

|  |  |
| --- | --- |
| **Step** | **Information** |
| 1 | A class is a blueprint that defines the structure and behavior of an object. Class contain attributes (data) and methods (functions) that relate to those objects. Object, on the other hand, are concrete instances of a class, have real values for attributes and are capable of executing methods defined in the class. In PHP, you can create a class with the class keyword and then create an object from that class with the new keyword. Here is a simple example: |
| 2 | Create a folder dasarWeb/JS12\_OOP and a new file called oop.php inside the folder. |
| 3 | Type this code into the oop.php file. |
| 4 |  |
| 5 | What do you understand from the code above. Record below your understanding. (Question No 1.1)   **Property**:   * $brand: A public property that holds the brand of the car (e.g., "Toyota" or "Honda").    **Method**:   * startEngine(): A public method that, when called, outputs the message "Engine started!".    **Object Creation**:   * Two objects of the Car class are created: $car1 and $car2. * The brand property of $car1 is set to "Toyota" and the brand property of $car2 is set to "Honda".    **Method and Property Access**:   * The startEngine() method is called on the $car1 object, printing "Engine started!". * The brand property of $car2 is accessed and echoed, printing "Honda". |
| 6 | Inheritance is one of the basic concepts in object-oriented programming (OOP) that allows a class to inherit properties and methods from other classes. An inherited class is called a subclass or child class, while a class that provides inheritance is called a superclass or parent class. This concept allows us to reuse code, extend functionality, and build class hierarchies.  The following is a simple example of the implementation of inheritance in PHP:  What do you understand from the code above. Record below your understanding. (Question No 1.2)   1. **Class Animal**:    * It has a protected property $name, which is intended to store the name of the animal.    * The constructor \_\_construct($name) is intended to initialize the animal's name, but it has a syntax error. It should use $this->name = $name; instead of $this->name $name;.    * It has two methods:      + eat(): This method outputs a message stating that the animal is eating.      + sleep(): This method outputs a message stating that the animal is sleeping, but it is missing proper syntax and braces. 2. **Class Cat (extends Animal)**:    * It inherits from the Animal class.    * The meow() method is intended to print a message saying the cat is meowing, but it has a syntax error. It should be written as echo $this->name . " says meow!<br>";. 3. **Class Dog (extends Animal)**:    * It inherits from the Animal class.    * The bark() method is intended to print a message saying the dog is barking, but again, it has syntax errors and should be corrected as echo $this->name . " says woof!<br>";. 4. **Object Creation**:    * Two objects are created: $cat of class Cat with the name "Whiskers" and $dog of class Dog with the name "Buddy".    * The following methods are called:      + $cat->eat(): Outputs "Whiskers is eating."      + $dog->sleep(): Outputs "Buddy is sleeping."      + $cat->meow(): Outputs "Whiskers says meow!"      + $dog->bark(): Outputs "Buddy says woof!" |
| 7 | Polymorphism is a concept in object-oriented programming that allows objects of different classes to respond to method calls in the same way. This can be realized in PHP through the use of interfaces and the use of overriding methods. With polymorphism, you can treat objects of different classes in a uniform way.  Here is a simple example of using polymorphism in PHP using the interface:  What do you understand from the code above. Record below your understanding. (Question No 1.3)   **Interface Shape**:   * The interface Shape defines one method: calculateArea(), which is intended to calculate the area of a shape. Any class that implements this interface must provide its own implementation of the calculateArea() method.    **Class Circle**:   * The Circle class implements the Shape interface, meaning it must define the calculateArea() method. * It has a private property $radius, which is set through the constructor. * The calculateArea() method calculates the area of the circle using the formula: π×radius2\pi \times \text{radius}^2π×radius2, but the multiplication operator is incorrectly written as ✶ instead of \*. * Correct formula: return pi() \* pow($this->radius, 2);    **Class Rectangle**:   * The Rectangle class also implements the Shape interface, so it must define the calculateArea() method. * It has two private properties: $width and $height, which are set through the constructor. * The calculateArea() method calculates the area of the rectangle, but the multiplication operator is missing. The correct formula should be: return $this->width \* $this->height;.    **Function printArea(Shape $shape)**:   * The function printArea() takes a parameter that must be an object implementing the Shape interface (i.e., it can accept instances of Circle, Rectangle, or any other class that implements Shape). * The function attempts to echo the area of the shape by calling $shape->calculateArea(). However, there is a syntax error in the echo statement. The proper syntax should be: echo "Area: " . $shape->calculateArea() . "<br>";.    **Object Creation**:   * A Circle object is created with a radius of 5, and a Rectangle object is created with a width of 4 and height of 6. * The printArea() function is called for both the Circle and the Rectangle objects, printing the areas. |
| 8 | Encapsulation is one of the concepts in object-oriented programming (OOP) that allows encapsulation of properties and methods in a class so that access to them can be controlled. This can help in applying access management principles and ensure that properties and methods that may change in the future do not compromise the integrity of the class or program as a whole.  Here is a simple example of encapsulation in PHP:  What do you understand from the code above. Record below your understanding. (Question No 1.4)   **Class Car**:   * **Private Properties**:   + $model: Holds the model of the car (e.g., "Toyota").   + $color: Holds the color of the car (e.g., "Blue"). * **Constructor \_\_construct($model, $color)**:   + This method initializes the model and color properties when a new object of the Car class is created. * **Getter Method getModel()**:   + Returns the value of the model property. * **Setter Method setColor($color)**:   + Accepts a new color as an argument and updates the color property with the new value. * **Getter Method getColor()**:   + Returns the value of the color property.    **Object Creation and Method Usage**:   * A Car object is created with the model "Toyota" and color "Blue" using the constructor. * The getModel() method is called to print the car's model, which outputs "Toyota". * The getColor() method is called to print the car's color, which initially outputs "Blue". * The setColor("Red") method is called to update the car's color to "Red". * The getColor() method is called again to print the updated color, which now outputs "Red". |
| 9 | Abstraction is one of the basic concepts in object-oriented programming (OOP) that allows you to hide internal details and expose only the necessary functionality. It helps in creating classes and methods that are general and flexible, allowing users to interact with objects without needing to know their internal implementations.  Here's a simple example of abstraction in PHP using abstract classes and methods:  What do you understand from the code above. Record below your understanding. (Question No 1.5)     **Abstract Class Shape**:   * This class defines an abstract method calculateArea(), which must be implemented by any class that extends Shape. * Abstract classes cannot be instantiated directly; they must be extended by concrete classes that provide implementations for the abstract methods.    **Class Circle (extends Shape)**:   * The Circle class extends the Shape class and provides a specific implementation of the calculateArea() method. * It has a private property $radius, which is set through the constructor. * The calculateArea() method computes the area of the circle using the formula π×radius2\pi \times \text{radius}^2π×radius2.    **Class Rectangle (extends Shape)**:   * The Rectangle class extends the Shape class and provides a specific implementation of the calculateArea() method. * It has two private properties: $width and $height. * The constructor is intended to initialize the width and height, but there is a syntax issue. The assignment to $width and $height should be inside the constructor's curly braces. * The calculateArea() method computes the area of the rectangle, but there is a syntax error in the multiplication. It should be: return $this->width \* $this->height;.    **Object Creation and Method Calls**:   * A Circle object is created with a radius of 5, but the instantiation syntax is incorrect. The correct syntax is: $circle = new Circle(5);. * A Rectangle object is created with a width of 4 and height of 6. * The areas of the circle and rectangle are printed, but there are syntax errors in the echo statements. |
| 10 | An interface is a concept in object-oriented programming that allows the definition of a contract or framework that the classes that implement it must follow. Interfaces don't have their own implementations, but only provide a declaration of methods and properties that the classes that use them must implement. This makes it possible to achieve polymorphism without requiring a single inheritance, so that a class can implement multiple interfaces.  Here is an example of using the interface in PHP:  What do you understand from the code above. Record below your understanding. (Question No 1.6)     **Interface Shape**:   * The Shape interface defines a method calculateArea() that any class implementing this interface must implement. The calculateArea() method is expected to compute the area of the shape.    **Interface Color**:   * The Color interface defines a method getColor() that any class implementing this interface must implement. The getColor() method is expected to return the color of the object.    **Class Circle (implements both Shape and Color)**:   * The Circle class implements both the Shape and Color interfaces, meaning it must define both the calculateArea() and getColor() methods. * The class has two private properties: $radius (for the radius of the circle) and $color (for the color of the circle). * The constructor initializes these two properties. * The calculateArea() method computes the area of the circle using the formula π×radius2\pi \times \text{radius}^2π×radius2. * The getColor() method returns the color of the circle.    **Object Creation and Method Calls**:   * An instance of Circle is created with a radius of 5 and color "Blue". * The calculateArea() method is called to compute and display the area of the circle. * The getColor() method is called to display the color of the circle. |
| 11 | Constructors and destructors are special methods in object-oriented programming (OOP) used in PHP to initialize and clean objects. A constructor is a method that is called automatically when a new object is created, whereas a destructor is a method that is called automatically when an object is deleted or no longer in use.  **Constructor**  The constructor uses \_\_construct special names in PHP. This constructor will be called automatically whenever a new object is created from a class that contains that constructor.  **Destructor**  The destructor uses \_\_destruct special names in PHP. This destructor will be called automatically when the object is deleted or the program finishes executing.  Here are examples of constructors and destructors:    What do you understand from the code above. Record below your understanding. (Question No 1.7)   **Class Car**:   * **Private Property $brand**: This property holds the brand of the car. * **Constructor (\_\_construct)**: The constructor takes one parameter ($brand) and assigns it to the $brand property. It also includes an echo statement to print "A new car is created" whenever a new instance of the class is created. * **Method getBrand()**: This method returns the value of the $brand property, allowing access to the car's brand. * **Destructor (\_\_destruct)**: The destructor is a special method that is automatically called when an object is destroyed (i.e., when it goes out of scope or the script finishes). It includes an echo statement to print "The car is destroyed" when the object is destroyed.    **Object Creation**:   * An object $car of the Car class is created with the brand "Toyota". * The constructor is called, which prints "A new car is created." and assigns "Toyota" to the $brand property. * The getBrand() method is called to retrieve the car's brand, which is then printed.    **Destructor**:   * When the script ends and the $car object goes out of scope, the destructor is called automatically, printing "The car is destroyed". |
| 12 | Encapsulation and Access Modifiers Encapsulation is one of the key concepts in object-oriented programming (OOP), and it involves wrapping data (variables) and methods (functions) in a class. This helps in hiding the internal implementation of a class and exposing only the necessary functionality. Access modifiers are a subset of encapsulation that allows you to control the level of access to properties and methods in a class.  PHP has three main access modifiers that can be used in classes:  Public: Properties or methods that are declared public can be accessed from outside the class, so they are open to access from anywhere.  Protected: Properties or methods that are declared protected can only be accessed from within the class itself and from its child classes (inheritance).  Private: Properties or methods that are declared private can only be accessed from within the class itself. They cannot be accessed from outside the class, not even by its child classes.  Here is an example of using access modifiers in PHP:  What do you understand from the code above. Record below your understanding. (Question No 1.8)     **Class Animal**:   * **Properties**:   + $name (public): This property is accessible from anywhere (outside the class, within the class, and from subclasses).   + $age (protected): This property is accessible within the class and by subclasses (inherited classes) but not from outside the class.   + $color (private): This property is only accessible within the class itself, not from outside or from subclasses. * **Constructor (\_\_construct)**: This method initializes the properties $name, $age, and $color with values provided when the object is instantiated. * **Methods**:   + getName(): This is a public method that returns the value of the $name property.   + getAge(): This is a protected method that returns the value of the $age property, but can only be accessed within the class or from a subclass.   + getColor(): This is a private method that returns the value of the $color property and can only be accessed within the Animal class itself.    **Object Creation**:   * An object $animal of the Animal class is created with the values "Dog" for name, 3 for age, and "Brown" for color. * The constructor is called, which initializes the properties with these values.    **Method Access**:   * The code attempts to print the name, age, and color of the animal. * The public $name property can be accessed directly outside the class using $animal->name. * The protected getAge() and private getColor() methods are not directly accessible from outside the class, which will lead to an error when trying to call them outside the class. |

**Practicum 2. CRUD with OOP**

|  |  |
| --- | --- |
| **Step** | **Information** |
| 1 | Create a new file dasarWeb/JS12\_OOP/database.php. Type the code as below. |
| 2 |  |
| 3 | Create a new file dasarWeb/JS12\_OOP/crud.php. Type the code as below. |
| 4 |  |
| 5 | Create a new file dasarWeb/JS12\_OOP/index.php. Type the code as below. |
| 6 |  |
| 7 | Create a new file dasarWeb/JS12\_OOP/edit.php. Type the code as below. |
| 8 | Run the code in practicum 2. What do you understand from the code above. Record below your understanding. (Question No 2.1)    **1. Database Connection (Database.php):**   * **Purpose**: Manages the connection to the database using MySQLi. * **Key Components**:   + The Database class contains private properties to store database connection parameters (host, username, password, and database name).   + The constructor (\_\_construct()) creates a new connection to the MySQL database using mysqli, and checks for connection errors. If the connection fails, the script will stop execution and display an error message.   **2. CRUD Operations (Crud.php):**   * **Purpose**: Provides methods for performing Create, Read, Update, and Delete (CRUD) operations on the jabatan (job title) table in the database. * **Key Methods**:   + **create()**: Inserts a new record into the jabatan table using the jabatan (job title) and keterangan (description) parameters.   + **read()**: Retrieves all records from the jabatan table and returns them as an array.   + **readById()**: Retrieves a single record from the jabatan table by its id.   + **update()**: Updates an existing record in the jabatan table with new values for jabatan and keterangan, based on the id.   + **delete()**: Deletes a record from the jabatan table based on its id.   **3. Edit Page (edit.php):**   * **Purpose**: Displays a form for editing an existing job title record, and updates the record when the form is submitted. * **Key Components**:   + The Crud class is instantiated and used to fetch the record to be edited ($tampil = $crud->readById($id);).   + The form is pre-filled with the current values of the jabatan and keterangan fields.   + Upon submission (POST request), the update() method is called to update the record, and the user is redirected back to the index page (header("Location: index.php");).   **4. Index Page (index.php):**   * **Purpose**: Displays all job title records in a table and allows users to add new records, as well as edit or delete existing ones. * **Key Components**:   + A form is provided for adding new records, which is shown in a Bootstrap modal.   + Upon submission of the form (POST request), the create() method is called to insert a new job title into the database.   + A table displays all records from the jabatan table. For each record, there are two action buttons:     - **Edit**: A link that leads to the edit.php page with the corresponding id to edit the record.     - **Delete**: A link that triggers the delete() method by passing the id of the record to be deleted via the URL (index.php?action=delete&id=<id>).   **5. OOP Example (oop.php):**   * **Purpose**: Demonstrates basic Object-Oriented Programming (OOP) concepts in PHP with a simple Car class. * **Key Components**:   + The Car class has a **public property** $brand and a **public method** startEngine().   + Two objects ($car1 and $car2) are created from the Car class, and their brand properties are set to "Toyota" and "Honda", respectively.   + The startEngine() method is called on $car1, printing "Engine started!".   + The brand property of $car2 is echoed, printing "Honda". |

https://github.com/Samidkun/DASARWEB.git