Midterm Examinations
First Semester
A)Y. 2024-2025

Course Number: **ITE601**
Course Title**: OBJECT ORIENTED PROGRAMMING**
Instructor: **JASON V. CASTELLANO**

GENERAL INSTRUCTIONS: This examination is designed to assess what you have learned; hence, work on each item by yourself within a time allotment of **120 minutes** or **2 hours**. Shade/ Write the letter that corresponds to the correct answer in your yellow booklet. Write your letter answer clearly and **no erasure**. **Any erasure** in your answer in **multiple choices** will mark it **wrong**. In the essay section erasure shall allow in your explanation. Take your time in answering each question and good luck class. Total number of items **80 points** including the essay questions.

Multiple Choice (80 items – 1 Point Each = 80 Points)

1. **What is an abstract class?**

   A) A class that cannot be instantiated on its own and serves as a blueprint for other classes.

   B) A class that can be instantiated directly.

   C) A class with no defined methods or properties.

   D) A class that can only be used in specific applications.

2. **Which programming principle is reinforced by using abstract classes?**

   A) Code reusability.

   B) Direct instantiation.

   C) Single responsibility principle.

   D) Automated inheritance.

3. **In object-oriented programming, an abstract class is primarily used to...**

   A) Define methods to be implemented by subclasses.

   B) Create instances directly.

   C) Hold private data exclusively.

   D) Prevent inheritance.

4. **What is the main reason for creating abstract classes in a program?**

   A) To enforce a standard structure among related classes.

   B) To avoid using interfaces.

   C) To simplify complex algorithms.

   D) To prevent inheritance from other classes.

5. **Why is code reusability associated with abstract classes?**

   A) Because they define common methods and properties that can be inherited by subclasses.

   B) Because they avoid memory usage in subclasses.

   C) Because they prevent classes from being instantiateD)

   D) Because they work with private properties exclusively.

6. **Encapsulation of shared logic in abstract classes means...**

   A) Common behavior is shared across subclasses while specific behaviors can be implemented as needeD)

   B) All methods are shared without exception in subclasses.

   C) No additional subclass behavior is alloweD)

   D) Each subclass ignores the shared methods.

7. **When creating an abstract class in code, which keyword is typically used?**

   A) abstract

   B) new

   C) static

   D) final

8. **If a subclass does not implement all abstract methods of an abstract class, what will occur?**

   A) A compilation error will occur.

   B) The program will ignore the methods.

   C) The subclass will compile successfully.

   D) The abstract methods will be skipped in execution.

9. **Which of the following scenarios best fits using an abstract class over an interface?**

   A) When shared code needs to be included in several related subclasses.

   B) When no method implementation is requireD)

   C) When classes need unique behaviors only.

   D) When direct instantiation is requireD)

10. **When using an abstract class, how does it enforce method implementation in subclasses?**

    A) By defining abstract methods that subclasses must implement.

    B) By creating methods that are private to subclasses.

    C) By restricting all subclasses from adding methods.

    D) By hiding methods from subclasses.

11. **What happens when an abstract class contains implemented methods?**

    A) Subclasses inherit the implemented methods directly.

    B) Subclasses must re-implement all methods.

    C) Abstract classes cannot contain implemented methods.

    D) Implemented methods in abstract classes are ignoreD)

12. **To prevent a class from being instantiated directly but allow it to be subclassed, one shoulD)..**

    A) Declare it as an abstract class.

    B) Make it a private class.

    C) Use a static class.

    D) Avoid using constructors.

13. **What will occur if a concrete subclass does not implement all abstract methods of its abstract superclass?**

A) Compilation error.

B) Successful compilation without warnings.

C) Run-time error only.

D) It will bypass unimplemented methods.

14. **In an inheritance hierarchy, an abstract class can help by...**

    A) Enforcing method signatures across multiple subclasses.

    B) Avoiding any method sharing across subclasses.

    C) Implementing every possible method directly.

    D) Making subclasses unique without shared functionality.

15. **When extending an abstract class, a subclass...**

    A) Must implement any abstract methods in the superclass.

    B) Can ignore all superclass methods.

    C) Needs to define no new methods.

    D) Must be marked abstract itself. **The benefit of encapsulating logic within an abstract class is to...**

    A) Share functionality while allowing subclass-specific behavior.

    B) Hide all functionality from subclasses.

    C) Ensure all subclasses have the same behavior.

    D) Restrict subclassing altogether.

16. **How does an abstract class enforce a structure for related classes?**

    A) By defining methods that must be implemented by each subclass.

    B) By allowing subclasses to implement any methods they choose.

    C) By avoiding any enforced structure.

    D) By creating private methods exclusively.

17. **If a developer creates a class that should not be instantiated but should contain shared methods, they shoulD)..**

    A) Make it an abstract class.

    B) Declare it as final.

    C) Avoid adding methods.

    D) Mark all methods as private.

18. **In analyzing class hierarchies, why might an abstract class be more beneficial than an interface?**

    A) Because it allows shared logic to be implemented once and inheriteD)

    B) Because it restricts instantiation.

    C) Because it avoids inheritance altogether.

    D) Because it forces unique methods in every subclass.

19. **Given an abstract class with multiple concrete subclasses, what happens if the superclass changes?**

    A) Subclasses may need to adapt to the changes.

    B) Subclasses are unaffecteD)

    C) Subclasses cease to exist.

    D) Superclass logic does not apply to subclasses.

20. **Why is encapsulating logic in an abstract class preferable over repeating code in multiple classes?**
    A) It centralizes shared behavior, reducing redundancy.
    B) It allows every class to have unique implementations.
    C) It prevents method inheritance.
    D) It limits class functionality.
21. **If two subclasses inherit an abstract method from a superclass, how do they differ?**
    A) Each subclass provides its own specific implementation of the methoD)
    B) They share the same method without changes.
    C) They ignore the inherited methoD)
    D) They merge methods together.
22. **Which scenario justifies using an abstract class in designing an application?**
    A) When multiple classes require shared, partially implemented methods.
    B) When no shared logic exists between classes.
    C) When each class requires unique behavior only.
    D) When instantiation is mandatory for all classes.
23. **How might using abstract classes impact application maintenance?**
    A) It can simplify maintenance by centralizing shared functionality.
    B) It complicates maintenance due to hidden methods.
    C) It reduces flexibility in subclassing.
    D) It forces constant subclass changes.
24. **What is an interface?**
    A) A blueprint for classes that defines a contract of methods that must be implementeD)
    B) A class that provides full method implementations.
    C) A subclass with additional properties.
    D) A method that can be overridden.
25. **By default, methods declared in interfaces are...**
    A) PubliC)
    B) Private.
    C) ProtecteD)
    D) Package-private.
26. **What is a primary characteristic of an interface?**
    A) It only provides method signatures, without implementation details.
    B) It allows for partial method implementation.
    C) It includes only private methods.
    D) It automatically implements methods.
27. **Why do classes implement interfaces?**
    A) To provide concrete implementations for the specified methods.
    B) To allow for private methods in subclasses.
    C) To inherit properties directly.
    D) To prevent other classes from accessing them.

28. **How does an interface enforce a "contract" in programming?**

    A) By requiring implementing classes to provide specific method implementations.

    B) By limiting access to protected members only.

    C) By sharing private properties.

    D) By hiding methods from subclasses.

29. Why might interfaces be preferred over abstract classes for certain designs?

    A) They support multiple inheritance.

    B) They enforce single inheritance only.

    C) They allow method implementations directly.

    D) They require private methods exclusively

30. When a class implements an interface, it must...

    A) Implement all of the interface's methods.

    B) Override only some methods.

    C) Inherit all properties without implementing.

    D) Ignore certain methods if they are not needeD)

31. **To use an interface in a class, which keyword is typically used in its declaration?**

    A) implements

    B) extends

    C) inherits

    D) defines

32. Which of the following statements about interfaces is correct?

    A) They allow classes to support multiple behaviors through multiple inheritance.

    B) They prevent classes from using inheritance.

    C) They provide full implementations for all methods.

    D) They allow for method bodies in the interface itself.

33. If a class implements two interfaces, each with a method printDetails(), what must the class do?

    A) Provide a concrete implementation for printDetails() once.

    B) Only implement printDetails() from one interface.

    C) Implement one and ignore the other.

    D) Avoid implementing either methoD)

34. What advantage does using an interface provide when designing a system with multiple unrelated behaviors?

    A) It allows classes to implement multiple interfaces, supporting multiple behaviors.

    B) It restricts classes to a single inheritance model.

    C) It forces the use of the abstract keyworD)

    D) It limits behavior to a single class only.

35. In Java, how can a class support both a Movable and Printable behavior simultaneously?

    A) By implementing both Movable and Printable interfaces.

    B) By inheriting from Movable and Printable.

C) By using a static keyword in the class.

D) By creating an abstract class with both behaviors.

36. Which of the following best describes a scenario where interfaces are beneficial?

A) When multiple unrelated classes need to follow a common protocol.

B) When classes need private methods exclusively.

C) When only one subclass is alloweD)

D) When complex hierarchy is unnecessary.

37. How does an interface differ from an abstract class in terms of functionality?

A) Interfaces do not provide any method implementation, whereas abstract classes can.

B) Interfaces are instantiated directly, unlike abstract classes.

C) Interfaces contain private datA)

D) Interfaces have no method declarations.

38. What keyword allows a class to provide multiple implementations for unrelated behaviors?

A) implements

B) extends

C) abstract

D) inherits

39. How does an interface help in ensuring class design flexibility?

A) By allowing multiple inheritance, providing options for diverse behaviors.

B) By restricting class methods to one interface.

C) By inheriting from only one class.

D) By making methods inaccessible.

40. Why might a developer choose an interface to design a system for managing multiple device types?

A) Interfaces allow each device to implement the necessary methods without inheriting properties.

B) Interfaces prevent method overrides.

C) Interfaces contain private implementations.

D) Interfaces limit the number of classes that can implement them.

41. When implementing an interface, a class...

A) Provides specific implementations for all methods defined in the interface.

B) Can skip methods that are not relevant.

C) Automatically inherits the method implementations.

D) Implements only public properties.

42. If two interfaces contain the same method name but with different implementations, how does a class handle this?

A) By defining a single implementation in the class.

B) By using both implementations directly.

C) By ignoring one of the implementations.

D) By creating an abstract methoD)

43. In designing a multimedia application, why might Playable be defined as an interface?

A) To allow various media types to implement specific playback methods.

B) To enforce playback in a single format only.

C) To avoid using abstract classes.

D) To prevent inheritance of Playable.

44. What is the role of an interface in defining a "contract" for implementing classes?

A) It requires classes to implement specific methods as defineD)

B) It limits methods to private access.

C) It allows methods to have default implementations.

D) It prevents subclassing altogether.

45. **Why might an interface be more advantageous than an abstract class when designing a plug-and-play architecture?**

A) Interfaces allow for multiple inheritance and diverse behavior additions.

B) Interfaces prevent shared logic across classes.

C) Interfaces allow private methods.

D) Interfaces restrict implementation.

46. When analyzing the difference between abstract classes and interfaces, which of the following is unique to interfaces?

A) They support multiple inheritance by allowing classes to implement multiple interfaces.

B) They provide shared functionality directly to subclasses.

C) They include private data members.

D) They prevent method overriding.

47. How does implementing an interface affect code reusability in large systems?

A) It enhances flexibility by allowing multiple unrelated behaviors in classes.

B) It restricts the use of common behaviors.

C) It simplifies inheritance only for a single class.

D) It requires that methods be protecteD)

48. When deciding between an interface and an abstract class, a developer should consider using an interface if...

A) The class needs to implement multiple unrelated behaviors.

B) The class needs shared functionality.

C) The class should have no public methods.

D) The class will not be inheriteD)

49. Why might interfaces be preferred in scenarios where multiple classes must adhere to the same method names but vary in implementation?

A) They enforce method implementation without providing shared logiC)

B) They prevent the addition of extra methods.

C) They provide private logiC)

D) They enforce a single class type.

50. In designing a new framework, when is an interface preferable over an abstract class?

A) When classes need to implement different behaviors while sharing the same method names.

B) When a single class structure needs shared functionality.

C) When inheriting protected methods.

D) When restricting behavior inheritance is necessary.

51. What impact does using interfaces have on system extensibility?

A) It improves extensibility by allowing classes to implement multiple behaviors flexibly.

B) It complicates code maintenance by enforcing single inheritance.

C) It reduces flexibility by enforcing protected access.

D) It restricts multiple classes from implementing the same methods.

52. What are design patterns in software development?

A) Standardized, reusable solutions to common software design problems.

B) A specific programming language syntax.

C) Debugging techniques for resolving errors.

D) Unique algorithms for optimizing performance.

53. Who introduced the concept of design patterns in software development?

A) The "Gang of Four

B) The founders of PHP.

C) Developers of object-oriented programming.

D) The inventors of Agile methodology.

54. Why are design patterns considered important in software development?

A) They promote code reusability and enhance maintainability and flexibility.

B) They are only applicable in PHP programming.

C) They focus solely on improving software security.

D) They make code incompatible with other frameworks.

55. What role do behavioral patterns play in design patterns?

A) Define communication between objects, focusing on interaction and responsibility.

B) Manage the lifecycle of objects.

C) Provide solutions for data storage.

D) Enforce strict coding guidelines.

56. If a developer needs a pattern that focuses on object creation, they should use...

A) A creational pattern.

B) A structural pattern.

C) A behavioral pattern.

D) A static pattern.

57. When a design pattern aims to manage complex relationships between classes and objects, it is typically...

A) A structural pattern.

B) A creational pattern.

C) A behavioral pattern.

D) A singleton pattern.

58. Which design pattern type is best for defining interactions and responsibilities between objects?

A) Behavioral pattern.

B) Structural pattern.

C) Creational pattern.

D) Modular pattern.

59. In developing a system where classes need to be organized to form larger structures, a developer should use...

A) Structural patterns.

B) Creational patterns.

C) Behavioral patterns.

D) Functional patterns.

60. A developer is working on a PHP application and wants to avoid repetitive coding tasks by applying a standardized solution. What should they consider using?

A) Design patterns.

B) Unique algorithms.

C) Custom functions.

D) Randomized procedures.

61. If a developer wants to control the lifecycle of objects in their application, which type of design pattern would be suitable?

A) Creational pattern.

B) Behavioral pattern.

C) Structural pattern.

D) Operational pattern.

62. To reduce the likelihood of errors in software development, developers use design patterns because...

A) They are tried and tested solutions.

B) They are only applicable to web applications.

C) They create unique, untested solutions.

D) They require constant modification.

63. When applying a creational pattern, a developer is primarily concerned with...

A) Managing object creation processes.

B) Simplifying database connections.

C) Enhancing the interaction between objects.

D) Creating database schemas.

64. What is a key benefit of using structural patterns in object-oriented design?

A) They enable effective composition of classes and objects.

B) They manage the timing of object creation.

C) They enforce singleton behavior.

D) They secure database operations.

65. To ensure a PHP application is flexible and easy to maintain, which of the following is recommended?

A) Applying appropriate design patterns.

B) Using as few classes as possible.

C) Avoiding object-oriented principles.

D) Reducing code readability.

66. In solving a recurring problem of dependency management, a developer should consider...

A) A design pattern that addresses object composition.

B) A random solution for each instance.

C) Changing the programming language.

D) Implementing custom functions without structure.

67. When aiming to "not reinvent the wheel," developers can benefit from design patterns because they...

A) Offer pre-established solutions to common problems.

B) Are language-specific, only for PHP.

C) Restrict flexibility in design choices.

D) Remove the need for object-oriented principles.

68. In analyzing the difference between creational and structural patterns, which of the following is a unique focus of creational patterns?

A) Managing object lifecycle and instantiation processes.

B) Structuring classes to form larger applications.

C) Defining interactions between objects.

D) Enforcing class inheritance rules.

69. What distinguishes behavioral patterns from structural patterns?

A) Behavioral patterns focus on communication between objects, whereas structural patterns focus on the composition of objects.

B) Behavioral patterns enforce single inheritance, while structural patterns allow multiple inheritance.

C) Structural patterns eliminate the need for classes.

D) Behavioral patterns handle memory management only.

70. **Why might a developer use a structural pattern over a behavioral pattern in certain designs?**

A) To define relationships and arrangements of objects to form complex structures.

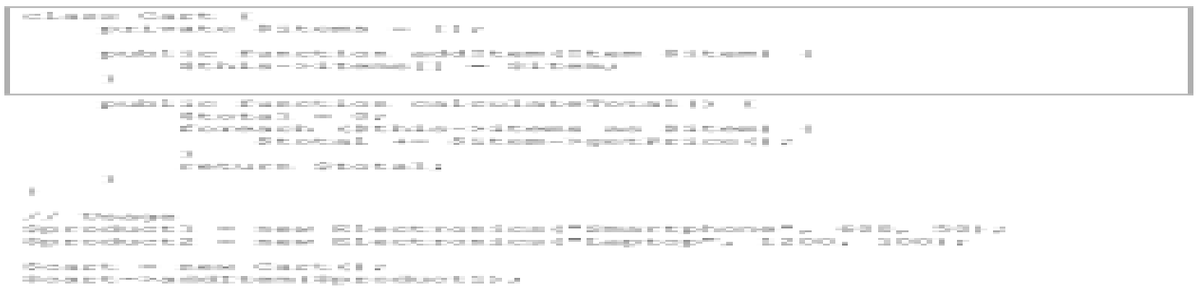
B) To enhance object communication exclusively.

C) To manage object lifecycle efficiently.

D) To simplify object instantiation.

Test II.

```php
<?php

interface Item {
    public function getName();
    public function getPrice();
}

abstract class Product implements Item {
    protected $name;
    protected $price;

    public function __construct($name, $price) {
        $this->name = $name;
        $this->price = $price;
    }

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

    public function getPrice() {
        return $this->price;
    }

    abstract public function getDiscountedPrice();
}

class Electronics extends Product {
    private $discount;

    public function __construct($name, $price, $discount) {
        parent::__construct($name, $price);
        $this->discount = $discount;
    }

    public function getDiscountedPrice() {
        return $this->price - $this->discount;
    }
}
```



71. **What is the role of the Item interface in the code?**

A) It defines methods that any class implementing it must use, ensuring consistency.

B) It is used to set default values for the Product class.

C) It allows the Cart class to automatically calculate total price.

D) It provides access to private properties of Product.

72. **What would happen if the getDiscountedPrice() method is not implemented in the Electronics class?**

A) PHP would throw a fatal error, as Electronics must implement the abstract methoD)

B) It would not affect functionality, as getDiscountedPrice() is optional.

C) Electronics will inherit getDiscountedPrice() automatically from Product.

D) The Cart class will throw an exception.

73. **Why is the Product class declared as abstract?**

A) It’s abstract because it provides incomplete functionality that subclasses must complete.

B) To prevent direct instantiation of any Product object.

C) To enforce stricter type checking in PHP.

D) So that Cart can automatically recognize subclasses of Product.

74. **Which of the following statements about interfaces in PHP is true?**

A) An interface can only define method signatures, not implementations.

B) An interface allows the definition of properties.

C) Interfaces in PHP can contain private methods.

D) Interfaces cannot be implemented by abstract classes.

75. **What would happen if Product was not marked as abstract?**

A) Nothing; Product could be instantiated directly, even without getDiscountedPrice().

B) PHP would throw a syntax error on execution.

C) Electronics would not be able to extend Product.

D) Product would still require an implementation for getDiscountedPrice().

76. **In PHP, how many interfaces can a single class implement?**

A) Any number of interfaces.

B) Only one interface.

C) No more than two interfaces.

D) It depends on the presence of abstract classes.

77. **Which of the following is true about abstract methods in PHP?**

A) They cannot contain a method body.

B) They can only be defined within interfaces.

C) They can be overridden with private access in child classes.

D) They automatically provide default functionality to subclasses.

78. **What would occur if getName() was not implemented in Electronics?**

A) No error, as Product already implements getName().

B) A runtime error, because each class must implement all interface methods.

C) Electronics would inherit a blank getName() methoD)

D) Cart would not be able to access getName() in Electronics.

79. **Why does Cart use addItem(Item $item) instead of addItem(Product $product)?**

A) This allows Cart to store any object that implements Item, not just Product.

B) It ensures that only Product subclasses can be addeD)

C) This method restricts Cart to storing only Electronics objects.

D) It forces all items to include a discount.

80. **If another class Furniture were to implement Item, what would be required for Furniture to comply?**

A) Implement both getName() and getPrice() methods.

B) Define a constructor that takes $name and $price.

C) Ensure that it extends Product.

D) Implement the getDiscountedPrice() methoD)