Mid Exam.pdf

Question Analysis

Question #2 (CLO-1):

1. **Question:** Create a Java class representing a Circle with a private radius attribute. Include public methods to set the radius and calculate the area of the circle.

•**Taxonomy Level Analysis:** This question can be classified under "Application" level as it requires students to apply knowledge of class creation and encapsulation in Java.

•**Match:** The intended level is correctly identified as it demands application of OOP concepts.

2. **Question:** Define a Java class Animal with a method makeSound() that prints "Animal makes a sound". Create a subclass Dog that overrides the makeSound() method to print "Dog barks".

•**Taxonomy Level Analysis:** This falls under the "Application" level since it also involves class creation and inheritance.

•**Match:** This matches the expected level as it tests students' practical skills in OOP.

3. **Question:** Create an interface Shape with a method calculateArea(). Implement this interface in classes Circle and Rectangle. Test polymorphism by calling calculateArea() on instances of both classes.

•**Taxonomy Level Analysis:** This again aligns with the "Application" level since it tests the understanding of interfaces and polymorphism.

•**Match:** Correctly matches the intended level.

4. **Question:** What is encapsulation and why is it important in OOP? How does encapsulation promote data hiding? Give an example of encapsulation in Java with a class containing private variables and public methods.

•**Taxonomy Level Analysis:** This question is at the "Understanding" level since it requires explanation and comprehension, not just application.

•**Mismatch:** The intended level is higher than specified.

•**Improvements Suggestions:**

•**Guidance**: Ask for a practical example implemented as code to demonstrate encapsulation.

•**Example Questions**:

1. Explain encapsulation in your own words and implement a Java example that demonstrates encapsulation.
2. Describe how encapsulation impacts software engineering principles with a practical coding example in Java.

5. **Question:** What is polymorphism and how does it contribute to code reusability? Explain the difference between compile-time polymorphism and runtime polymorphism. Give an example of polymorphism in Java using method overriding.

•**Taxonomy Level Analysis:** Similar to the previous question, it requires higher-level thinking and synthesis of information.

•**Mismatch:** The intended level is higher than specified.

•**Improvements Suggestions:**

•**Guidance**: Incorporate a practical coding task that demonstrates both compile-time and runtime polymorphism.

•**Example Questions**:

1. Create a Java program that shows how both compile-time and runtime polymorphism work.
2. Write an explanation of polymorphism with examples in Java that illustrate both types.

Question #3 (CLO-2):

1. **Question:** Identify the OOP concept(s) demonstrated in the Student class. How could you further extend this class to demonstrate inheritance?

•**Taxonomy Level Analysis:** This fits the "Analysis" level as it requires identification and evaluation of concepts.

•**Match:** This matches the expected level since it requires deeper conceptual understanding.

2. **Question:** You are tasked with developing a library management system for a local community library. The system should allow librarians to manage books, persons, and borrowing transactions efficiently.

•**Taxonomy Level Analysis:** This task fits the "Synthesis" level as it requires creating a full system based on given requirements.

•**Match:** Correctly designated as higher-order thinking.

Solution Analysis

1. Solution for Circle Class

•**Rating:** 8/10

•**Key Elements:** Correct use of private attributes and public methods to interact with the attribute.

•**Suggestions for Improvement:** Could introduce error handling (e.g., non-negative radius) to enhance robustness.

2. Solution for Animal Class and Subclass

•**Rating:** 9/10

•**Key Elements:** Correct implementation of inheritance and method overriding.

•**Suggestions for Improvement:** Include more examples or different animals for a more comprehensive design.

3. Solution for Shape Interface

•**Rating:** 8/10

•**Key Elements:** Proper use of interfaces and implementation in multiple classes.

•**Suggestions for Improvement:** Testing polymorphism could be more explicit within the solution (e.g., showing polymorphic behavior in action).

4. Solution for Encapsulation Explanation

•**Rating:** 7/10

•**Key Elements:** Good definition and importance described.

•**Suggestions for Improvement:** Request code examples to enhance the response, showing encapsulation in action.

5. Solution for Polymorphism Explanation

•**Rating:** 7/10

•**Key Elements:** Comprehensive understanding of the concept described.

•**Suggestions for Improvement:** Needs practical examples of compilation-time and run-time polymorphism in Java for clarity.

Detailed Feedback and Suggestions

For further improvement:

- **Encourage Practical Implementations**: More emphasis on demonstrating concepts via coding examples.
- **Clarification on Terms**: Terms such as encapsulation and polymorphism should be coupled with real-world analogies to aid understanding.
- **Augmenting Depth**: Implement follow-up tasks that require students to expand on initial answers as they gain deeper insights.

Overall, the exam questions show alignment with intended learning outcomes, but enhancements for certain areas can push them to higher taxonomy levels, thus fostering deeper learning experiences.
