# 1Z0-808 Exam Topic Reviewer

TopicId: 1019

Topic: Encapsulation and Access Modifiers

August 5, 2025

# Pillar 1: Encapsulation

Today we're tackling the first of the four great pillars of Object-Oriented Programming: **Encapsulation**. The concept is simple but powerful: we bundle the data (fields) and the methods that operate on that data into a single unit, the class. But encapsulation is more than just bundling. Its main goal is **data hiding**. We want to protect an object's internal state from being changed in unexpected or invalid ways by the outside world. We achieve this by hiding the implementation details and exposing only a controlled, public interface. Think of it like the dashboard of a car. It *encapsulates* the engine's complexity. You have a simple interface (a gas pedal), and you don't need to know—or mess with—the fuel injection system directly. This prevents you from accidentally breaking the engine.

## Implementing Encapsulation in Java

The standard strategy is straightforward:

(a) Declare all instance variables as `private`. This makes them inaccessible outside the class.

(b) Provide `public` methods, called **getters** (accessors) and **setters** (mutators), to read and modify the private fields.

```
public class Employee {
    private String name;
    private double salary;

    // Getter for name
    public String getName() {
        return name;
    }

    // Setter for name
    public void setName(String name) {
        if (name != null && !name.trim().isEmpty()) {
            this.name = name;
        }
    }

    // Getter for salary
    public double getSalary() {
        return salary;
    }

    // A setter can contain validation logic!
    public void setSalary(double salary) {
        if (salary >= 0) { // Protects the object's state
            this.salary = salary;
        }
    }
}
```

# 1 Java's Access Modifiers

Java uses four access modifiers to enforce encapsulation and control visibility. You must know these inside and out for the exam.

- `public`: The least restrictive. The member is accessible from any class in any package. This is for your public API.
- `protected`: The member is accessible within its own package, AND to subclasses that are in *different* packages. This is a common point of confusion.
- `default` (Package-Private): This is what you get if you specify **no modifier at all**. The member is accessible only to classes in the exact same package. Subclasses in a different package CANNOT access it.
- `private`: The most restrictive. The member is accessible only from within the same class file.

### The Definitive Access Modifier Table

Memorize this table. It's the key to dozens of potential exam questions.

| Modifier | Same Class | Same Package | Subclass (Diff. Pkg) | World (Diff. Pkg) |
|---|---|---|---|---|
| `public` | Yes | Yes | Yes | Yes |
| `protected` | Yes | Yes | Yes | No |
| `default` | Yes | Yes | No | No |
| `private` | Yes | No | No | No |

# 2 Exam Traps and Nuances

- **Top-Level Classes:** A class declaration that is not nested inside another class can only be `public` or `default`. It can never be `private` or `protected`. The exam might show this invalid code.
- **`protected` vs. `default`:** This is the trickiest comparison. A subclass in another package can access a `protected` member of its superclass, but not a `default` member. This is a favorite exam topic.
- **Method Overriding:** When a subclass overrides a method from its superclass, the access modifier in the subclass must be **the same or more accessible**. For example, you can override a `protected` method with a `public` one, but you cannot override a `public` method with a `protected` one.

# 3 Key Takeaways for the 1Z0-808 Exam

- Encapsulation means **data hiding**. Protect your fields by making them `private` and provide `public` getters and setters.

- Master the access modifier visibility table. Be able to recall it instantly.
- Pay special attention to the difference between `protected` and `default` access, especially in the context of inheritance across packages.
- Remember the rules for access modifiers on top-level classes and for overridden methods.