Encapsulation in Java with Example Programs

TechRounder PDF Edition

Live article: https://www.techrounder.com/development/encapsulation-in-java-with-example-programs/

By Vipin PG | Published November 8, 2023 | Updated January 8, 2026 | Format: Article | 3 min read

In brief

When learning Java, one of the fundamental concepts that every aspiring programmer must understand is encapsulation. Encapsulation in Java is a key concept in object-oriented programming, and it plays a crucial role in ensuring data integrity and code maintainability.

When learning Java, one of the fundamental concepts that every aspiring programmer must understand is encapsulation. Encapsulation in Java is a key concept in object-oriented programming, and it plays a crucial role in ensuring data integrity and code maintainability.

In this article, we will explore encapsulation, why it is important, and how to implement it in Java. We'll also provide example programs to illustrate the concept in action. Also, we'll explore the significance of Java Training.

What is Encapsulation?

One of the four core ideas of OOP (Object-Oriented Programming), along with inheritance, polymorphism, and abstraction, is encapsulation. It entails combining methods (functions) that manipulate the data with data (attributes) into a single unit known as a class. This unit provides the blueprint for making objects, or instances, of that class.

Encapsulation's main objectives are to conceal an object's internal workings and provide a regulated interface for communication. A common way to describe this idea is "data hiding." It enables you to limit access to the data and safeguard its integrity. You may guarantee that data is accessed and altered in a regulated and consistent way by encapsulating it.

Why is Encapsulation Important?

Encapsulation is a fundamental idea in Java programming since it provides several important advantages.

Data Protection: An object's internal state is shielded from outside intervention via encapsulation. It guarantees that an item stays in a proper state by preventing accidental changes to its data.

Code Maintainability: You may divide concerns by enclosing data and functions inside of classes. Your code becomes simpler to maintain and more modular as a result. Code that uses a class's public interface is unaffected by modifications to the class's internal implementation.

Flexibility: Changes made to a class's internal implementation may be made without impacting the code that utilizes it, thanks to encapsulation. For big software projects to evolve and be maintained, flexibility is essential.

Information Hiding: The outer world is shielded from the complexity of a class's implementation via encapsulation. This lets other developers using your class concentrate on the interface you've supplied and lessens their cognitive strain.

Security: When using encapsulation, you may impose validation tests and access limitations. You may, for instance, make sure that only authorized code has access to sensitive data.

Implementing Encapsulation in Java

In Java, access modifiers are the main tool used to establish encapsulation. To manage the visibility of classes, methods, and variables, Java has four different kinds of access modifiers: Package-private (default), protected, private, and public. You may designate who has access to and can change a class's components using these modifiers.

Any class may access elements that have the public modifier applied to them. They thus possess the most lenient access level.

Only elements in the same class may access elements with the private modifier. This offers the Subclasses and other elements in the same package that may access elements that have the protected modifier. Situations involving inheritance often employ this.

The element is regarded as package-private if no access modifier is given, which indicates that it may be accessed from within the same package but not from outside of it.

Let's examine a little example to show you how Java encapsulation functions:

The Person class in this example has the age and name of private data members. These values may be retrieved using the accessors getName and getAge and modified in a controlled manner with the mutators setName and setAge.

Only the specified methods may be used to access and modify the name and age properties, thanks to private access modifiers. The internal data is encapsulated in this way to prevent external abuse.

Example Programs

Now, let's look at a few sample programs to understand how encapsulation works in real-world scenarios.

Example 1: Using the Person Class

The name and age properties are created and retrieved in this program by creating an instance of the Person class and using its accessor methods. Encapsulation guarantees that the data is accessed and altered in compliance with established guidelines.

Example 2: Attempting to Modify Person Data

In this example, we try to specify an incorrect age of 5, but the Person class's encapsulation stops us. The age is still valid and unaltered.

Conclusion

A key idea in Java and object-oriented programming is encapsulation. In your software projects, it encourages information hiding, code maintainability, flexibility, data protection, and security. You may include encapsulation into your Java classes and ensure that data is accessed and updated in a controlled way by utilizing access modifiers.