Discussion 7: scope minimization in Java

Discussion Topic:

What are the benefits of scope minimization in Java?
Why is it a good technique to practice?
Provide an appropriate source code example to illustrate your points.

My Post:

Hello Class,

In Java, the scope of a variable is the part of a program where the variable can be accessed (Mahrsee, 2024). The scope can be a Class scope, method scope, or block scope. Java does not have global variables like C++ does; global variables are variables that can be accessed from anywhere in the program. In other words, the variables have a global scope.

Java inherently minimizes scope by encapsulating everything in classes. Java is a strictly object-oriented programming (OOP) language rather than a procedural one like C. On the other hand, C++ supports both paradigms, OPP and procedural programming. Anyhow, scope minimization is an approach with the goal of improved readability, better maintainability, and reduced chance of errors (Carter, 2021). DCL53-J SEI CERT Oracle Coding Standard for Java (CMU, n.d.) recommends minimizing the scope of variables to:

"avoid common programming errors, improves code readability by connecting the declaration and actual use of a variable, and improves maintainability because unused variables are more easily detected and removed. It may also allow objects to be recovered by the garbage collector more quickly, and it prevents violations of DCL51-J. Do not shadow or obscure identifiers in subscopes."

Minimizing the scope of a variable also adds a layer of security, as the variable is restricted to the context where it is needed. This reduces access, manipulation, or misuse by other parts of the program, limiting possible vulnerabilities. For example, in Java, declaring a class variable '*private*' would restrict its scope within the class, preventing other classes from directly modifying or accessing it. However, if the variable needs to be accessed or modified, it can only be done through controlled methods, such as *getters* or *setters*, which encapsulate the variable or return a copy of it; additionally, they can implement an extra layer or validation or logic that ensures that the variable is properly utilized.

Below is an example of how to apply scope minimization in Java can look like:

```
public class Employee {
    // Private class variables to restrict access
    private String name;
    private double salary;

    // Constructor to initialize Employee
    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Getter for name (read-only access)
    public String getName() {
        return name;
    }

    // Getter and setter for salary with validation
    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        if (salary > 0) {
            this.salary = salary; // Allow update if valid
        } else {
            throw new IllegalArgumentException("Salary must be greater than 0.");
        }
    }

    // Method to provide an increment with controlled logic
    public void applyBonus(double percentage) {
        if (percentage > 0 && percentage <= 20) {
            this.salary += this.salary * (percentage / 100);
        } else {
            throw new IllegalArgumentException("Bonus percentage must be between 0 and 
20.");
        }
    }

    // Display employee details
    public void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Salary: $" + salary);
    }
}
```

```
public class Main {
    public static void main(String[] args) {

       // Create an Employee object
        Employee emp = new Employee("Alice", 50000);
        System.out.println("Initial Salary:");
        emp.printDetails();

        // Modify Salary
        emp.setSalary(55000); // Update salary
        emp.applyBonus(10);   // Apply bonus
        System.out.println("\nUpdated Salary:");
        emp.printDetails();

        // Attempting to set an invalid salary
        System.out.println("\nInvalid salary (-10000):");
        try {
            emp.setSalary(-10000);
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Outputs:

```
Name: Alice
Salary: $50000.0

Updated Salary:
Name: Alice
Salary: $60500.0

Invalid salary (-10000):
Salary must be greater than 0.
```

To summarize, minimizing variable scope in Java code improves code readability, maintainability, and security by restricting access to where variables are needed most. Java is strictly object-oriented programming (OOP) language implying that it encapsulates data and variables within classes. This approach not only prevents unintended interactions and vulnerabilities but also aligns with best practices for efficient and secure programming.

References:

Carter, K. (2021, February 10). Effective Java: Minimize The Scope of Local Variables. DEV Community. https://dev.to/kylec32/effective-java-minimize-the-scope-of-local-variables-3e87

CMU — Software Engineering Institute (n.d.) DCL53-J. Minimize the scope of variables. *SEI CERT Oracle coding standard for Java.* Carnegie Mellon University. Software Engineering Institute.

Mahrsee, R. (2024, May 13). *Scope of variables in Java*. GeeksforGeeks. https://www.geeksforgeeks.org/variable-scope-in-java/