Coding Standards for the

National Disaster Response System

1. Naming Conventions

- **Classes and Interfaces**: Use nouns in PascalCase (capitalize each word).
  - **Example**: `CustomerManager`, `GuidelineProcessor`
- **Methods**: Use verbs in camelCase (start with lowercase).
  - **Example**: `calculateTotalAmount()`, `fetchGuidelineDetails()`
- **Variables**: Use meaningful camelCase names. Avoid one-letter or ambiguous names.
  - **Example**: `employeeName`, `totalAmount`
- **Constants**: Use uppercase letters with underscores between words.
  - **Example**: `MAX_RETRY_ATTEMPTS`, `DEFAULT_BUFFER_SIZE`

---

2. Declaration Order

Declare elements in the following order to enhance readability:

- **public ➔ protected ➔ private**
- Order methods, variables, and constructors following this structure so that the most accessible components appear first.

---

3. Curly Braces

- Use the inline style for constructors, methods, and control structures, placing the opening brace on the same line.
- **Example**:

```
java
Copy code
public GuidelineViewModel() {
    repository = GuidelineRepository.getInstance();
    guidelines = repository.getGuidelines();
}
```

- Avoid extra lines after opening or before closing braces.

---

4. Indentation

- Use 4 spaces per indentation level for consistency across environments.
- Avoid tabs.
- Ensure consistent indentation for blocks, control structures, and method declarations.
- **Example**:

```
public void processGuideline() {
    for (int i = 0; i < guidelines.size(); i++) {
        Guideline guideline = guidelines.get(i);
        // process guideline
    }
}
```

---

5. White Space

- Surround operators, keywords, and commas with spaces for readability.
  - **Example**: `int total = (a + b) * c;`
- Keep spaces between keywords like `if`, `while`, `for`, and parentheses.
  - **Example**: `if (isValid) { ... }`

---

6. Comments

- **Single-line Comments**: Use `//` for brief explanations.
  - **Example**: `// Increment guideline count`
- **Block Comments**: Use for larger explanations.
  - **Example**:

```
/*
 * This method processes a guideline and updates the database.
 * Ensure guidelines are validated before processing.
 */
```

- **Javadoc Comments**: Use for documenting classes, methods, and fields.
  - **Example**:

```
/**
 * Calculates the total number of guidelines.
 * @param guidelines List of guidelines.
 * @return Total count of guidelines.
 */
```

---

7. Method Length and Complexity

- Methods should be short and focused on a single task.
- Break down complex logic into smaller methods.
- **Example**:

```
public void processGuideline(Guideline guideline) {
    validateGuideline(guideline);
    updateDatabase(guideline);
}
```

---

8. Variable Scope

- Declare variables in the narrowest possible scope.
- Avoid global variables unless necessary.
- **Example**:

```
public void calculateTotal() {
    double total = 0.0;
    for (Item item : items) {
        total += item.getPrice();
    }
}
```

---

9. Lambda Expressions

- Use lambda expressions to simplify code but avoid over-complicating expressions.
- **Example**:

```
List<Integer> filteredList = list.stream()
                                 .filter(num -> num > 50)
                                 .collect(Collectors.toList());
```

---

10. Boxing and Unboxing

- Be cautious with autoboxing to avoid `NullPointerException` and performance issues.
- **Example**:

```
Integer count = null; // Avoid boxed types that may be null
```

---

11. Exception Handling

- Handle exceptions with specific `try-catch` blocks. Avoid catching generic `Exception`.
- **Example**:

```
try {
    Connection con = DriverManager.getConnection(DATABASE_URL,
USERNAME, PASSWORD);
    // process connection
} catch (SQLException e) {
    e.printStackTrace();
}
```

---

12. String Handling

- Use `StringBuilder` or `StringBuffer` for modifying strings to avoid unnecessary object creation.
- **Example**:

```
StringBuilder sb = new StringBuilder();
sb.append("Guideline ID: ").append(guidelineId).append(" processed.");
```

---

13. Testing

- Follow Test-Driven Development (TDD) principles. Use JUnit for unit tests, keeping them simple and focused.
- **Example**:

```
@Test
public void testCalculateTotalAmount() {
    Guideline guideline = new Guideline(...);
    double total = guideline.calculateTotalAmount();
    assertEquals(100.0, total, 0.01);
}
```

---

14. Interfaces and Abstraction

- Use interfaces to define contracts and prefer dependency injection.
- **Example**:

```
public interface Repository {
    void fetchData();
}
public class GuidelineRepository implements Repository {
    @Override
    public void fetchData() {
        // Implement fetch logic
    }
}
```

---

15. Immutability

- Make objects immutable where possible by using `final` fields and avoiding setters.
- **Example**:

```
public class Guideline {
    private final String id;
    private final List<Step> steps;

    public Guideline(String id, List<Step> steps) {
        this.id = id;
        this.steps = Collections.unmodifiableList(steps);
    }
}
```

---

16. Libraries and APIs

- Use standard Java libraries to maintain code reliability and consistency.
- **Example**:

```
import java.util.List;
import java.util.stream.Collectors;
```

References:

1. Se-Education: https://se-education.org/guides/conventions/java/intermediate.html
2. Developer.com: https://www.developer.com/design/top-10-java-coding-guidelines/
3. GeeksforGeeks: https://www.geeksforgeeks.org/coding-guidelines-in-java/
4. Javatpoint: https://www.javatpoint.com/coding-guidelines-in-java
5. Oracle: https://www.oracle.com/java/technologies/javase/codeconventions-introduction.html