Best Coding Practices & Industry Standards in Java

1. Code Readability & Formatting

✅ Follow Standard Naming Conventions

- Use meaningful and descriptive names.
- Class names: `PascalCase` (e.g., `PatientRecord`)
- Method names: `camelCase` (e.g., `calculateTotalAmount()`)
- Constants: `UPPER_CASE_SNAKE_CASE` (e.g., `MAX_LIMIT`)

✅ Maintain Proper Indentation & Formatting

- Use 4 spaces per indentation level.
- Keep lines short (ideally ≤ 80–100 characters).
- Use meaningful spacing for readability.

Example:

```java
public class PatientRecord {
    private String patientName;

    public void displayDetails() {
        System.out.println("Patient Name: " + patientName);
    }
}
```

Best Coding Practices & Industry Standards in Java

2. Use Proper Indentation & Formatting

✅ **Use 4 spaces per indentation level** (avoid tabs).
✅ **Keep lines within 80-100 characters** for readability.
✅ **Use braces `{}`** even for single-line `if` statements

```java
if (age > 18) {
    System.out.println("Eligible to vote.");
}
```

---

3. Write Small & Focused Methods

✅ A method should **perform one task only**.
✅ Keep **methods ≤ 20-30 lines** for clarity.
✅ **Avoid hardcoding values** – use constants.

Example:

```java
public double calculateDiscount(double price) {
    final double DISCOUNT_RATE = 0.10;
    return price * DISCOUNT_RATE;
}
```

---

4. Use Comments & Documentation

✅ Write **Javadoc comments** for classes and methods.
✅ Avoid **excessive** or **obvious** comments.

Example:

```java
/**
 * Calculates the total amount including tax.
 * @param price The base price of the item.
 * @param taxRate The tax rate percentage.
 * @return Total price after tax.
 */
public double calculateTotal(double price, double taxRate) {
    return price + (price * taxRate / 100);
}
```

---

5. Use Meaningful Conditions

✅ Avoid unnecessary comparisons.
✅ Use **boolean expressions** directly.

```java
//Bad Practice:
if (isAvailable == true) {   // Unnecessary comparison
    System.out.println("Available");
}

//Good Practice:
if (isAvailable) {    // Simpler condition
    System.out.println("Available");
}
```

Best Coding Practices & Industry Standards in Java

6. Use Proper Exception Handling

✅ Use **specific exceptions** (not just `Exception`).
✅ Handle errors gracefully.

Example:

```java
try {
    int result = 10 / 0; // This will cause an exception
} catch (ArithmeticException e) {
    System.out.println("Error: Division by zero is not allowed.");
}
```

---

7. Avoid Redundant Object Creation

✅ **Use existing objects** instead of creating new ones unnecessarily.

```java
//Bad Practice:
String name = new String("John");  // Unnecessary object creation

//Good Practice:
String name = "John";  // Efficient way
```

Best Coding Practices & Industry Standards in Java

8. Use StringBuilder Instead of String Concatenation

✅ **Use `StringBuilder` for repeated string modifications.**

```java
//Bad Practice:
String message = "Hello";
message += " World"; // Creates multiple unnecessary objects

//Good Practice:
StringBuilder sb = new StringBuilder("Hello");
sb.append(" World");
System.out.println(sb.toString());
```

---

9. Use Collections Properly

✅ **Use `ArrayList` instead of `LinkedList` for most cases.**
✅ **Use `HashMap` instead of `Hashtable` if thread safety is not required.**

Example:

```java
List<String> names = new ArrayList<>();
names.add("Alice");
names.add("Bob");
```

Best Coding Practices & Industry Standards in Java

10. Use Logging Instead of `System.out.println`

✅ **Use `SLF4J` or `Log4j`** instead of printing to console.

Example:

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyApp {
    private static final Logger logger = LoggerFactory.getLogger(MyApp.class);

    public static void main(String[] args) {
        logger.info("Application started successfully.");
    }
}
```

---

11. Keep Code DRY (Don’t Repeat Yourself)

✅ **Avoid duplicating code** – use methods or constants.

```java
//Bad Practice (Repeating Code):
double price1 = 100;
double price2 = 200;
double total1 = price1 + (price1 * 0.10);
double total2 = price2 + (price2 * 0.10);

//Good Practice (Using a Method):
public double calculateTotal(double price) {
    return price + (price * 0.10);
}
```

Best Coding Practices & Industry Standards in Java

12. Version Control & Code Reviews

✅ **Write meaningful commit messages.**
✅ **Use branches for new features.**

```
git commit -m "Fix: Improved error handling in PaymentService"
```

✅ Conduct Code Reviews

- Use tools like **SonarLint** and **Checkstyle** for static code analysis.
- Encourage peer reviews for better quality.

---

13. Concurrency & Thread Safety

✅ Use `ExecutorService` Instead of `Thread`

- Helps in managing a thread pool efficiently.

```
ExecutorService executor = Executors.newFixedThreadPool(10);
executor.submit(() -> System.out.println("Task executed"));
```

✅ Minimize Synchronization

- Use `volatile`, `AtomicInteger`, `ConcurrentHashMap` where necessary.
- Prefer immutable objects for thread safety.

Best Coding Practices & Industry Standards in Java

14. Object-Oriented Principles (OOP)

✅ Follow SOLID Principles

- **S**ingle Responsibility Principle (SRP) – Each class should do one thing.
- **O**pen/Closed Principle – Classes should be open for extension, but closed for modification.
- **L**iskov Substitution Principle – Subtypes should be replaceable for their base types.
- **I**nterface Segregation Principle – Don't force implementations to use unnecessary methods.
- **D**ependency Inversion Principle – Depend on abstractions, not concrete classes.

```java
public interface Payment {
    void processPayment();
}
public class CreditCardPayment implements Payment {
    @Override
    public void processPayment() {
        System.out.println("Processing Credit Card Payment...");
    }
}
```

Best Coding Practices & Industry Standards in Java

15. Code Efficiency & Performance

✅ Use Efficient Data Structures

- Prefer `ArrayList` over `LinkedList` in most cases.
- Use `HashMap` instead of `Hashtable` for non-thread-safe operations.
- Use `StringBuilder` instead of `String` for concatenations.

```
StringBuilder sb = new StringBuilder();

sb.append("Hello").append(" World");
```

✅ Avoid Redundant Object Creation

- Use `static` factory methods where possible.
- Reuse objects instead of creating new instances.

---

16. Testing & CI/CD Best Practices

✅ Write Unit Tests

- Use **JUnit** and **Mockito** for testing.
- Follow **AAA** pattern – Arrange, Act, Assert.

```java
@Test
public void testCalculateTotalAmount() {
    double result = service.calculateTotal(100, 5);
    assertEquals(105, result, 0.01);
}
```

✅ Use CI/CD for Automated Builds

- Integrate tools like **Jenkins, GitHub Actions, GitLab CI/CD**.
- Ensure test coverage in **SonarQube**.

---

17. Security Best Practices

✅ Validate Inputs & Use Prepared Statements

- Prevent SQL injection using parameterized queries.

```
String query = "SELECT * FROM users WHERE username = ?";

PreparedStatement stmt = conn.prepareStatement(query);

stmt.setString(1, username);
```

✅ Avoid Hardcoded Credentials

- Store secrets in environment variables or vaults.

Final Summary

Best Coding Practices & Industry Standards in Java

✅ Best Practice	🚀 Why It's Important
Follow naming conventions	Improves readability
Use indentation & formatting	Makes code clean
Keep methods short	Improves maintainability
Use meaningful conditions	Reduces confusion
Handle exceptions properly	Prevents crashes
Avoid redundant object creation	Saves memory
Use `StringBuilder` for concatenation	Improves performance
Use collections wisely	Optimizes data handling
Use logging instead of `System.out.println`	Better debugging
Keep code **DRY**	Reduces redundancy

Best Coding Practices & Industry Standards in Java

Code Coverage in Java using SonarQube and JUnit

🔹 What is Code Coverage?

Code coverage is a measure of how much of your application code is executed during testing. SonarQube helps analyze **code quality** and **coverage** by integrating with JUnit and JaCoCo.

---

1. Tools Required 🛠️

To measure code coverage using **SonarQube** and **JUnit**, you need:

✅ **SonarQube** – Code quality and security analysis.
✅ **JaCoCo** – Java Code Coverage (integrates with JUnit).
✅ **JUnit 5** – Unit testing framework.
✅ **Maven/Gradle** – Build automation tool.

---

2. Setup SonarQube for Code Coverage 🎯

🔹 Step 1: Install and Start SonarQube

If you haven't installed SonarQube:

🖥️ **Start SonarQube Locally (Using Docker)**

```
docker run -d --name sonarqube -p 9000:9000 sonarqube:lts
```

👉 SonarQube will be available at **http://localhost:9000** (default credentials: `admin/admin`).

Best Coding Practices & Industry Standards in Java

🔹 Step 2: Add JaCoCo to Your Java Project

For Maven (Add to `pom.xml`)

```java
<build>
    <plugins>
        <!-- JaCoCo Plugin for Code Coverage -->
        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.8</version>
            <executions>
                <execution>
                    <goals>
                        <goal>prepare-agent</goal>
                    </goals>
                </execution>
                <execution>
                    <id>report</id>
                    <phase>verify</phase>
                    <goals>
                        <goal>report</goal>
                    </goals>
                </execution>
            </executions>
        </plugin>

        <!-- SonarQube Plugin -->
        <plugin>
            <groupId>org.sonarsource.scanner.maven</groupId>
            <artifactId>sonar-maven-plugin</artifactId>
            <version>3.9.1.2184</version>
        </plugin>
    </plugins>
</build>
```

Best Coding Practices & Industry Standards in Java

🔹 Step 3: Configure SonarQube in `sonar-project.properties`

Create a `sonar-project.properties` file in the root of your project:

```
Java
sonar.projectKey=MyJavaProject
sonar.projectName=Java Code Coverage
sonar.host.url=http://localhost:9000
sonar.login=admin
sonar.password=admin
sonar.sources=src/main/java
sonar.tests=src/test/java
sonar.java.binaries=target/classes
sonar.coverage.jacoco.xmlReportPaths=target/site/jacoco/jacoco.xml
```

---

3. Write Java Code & JUnit Tests ✅

🔹 Java Class (Calculator.java)

```
Java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}
```

Best Coding Practices & Industry Standards in Java

- **JUnit Test Class (CalculatorTest.java)**

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorTest {
    @Test
    public void testAddition() {
        Calculator calculator = new Calculator();
        assertEquals(5, calculator.add(2, 3));
    }

    @Test
    public void testSubtraction() {
        Calculator calculator = new Calculator();
        assertEquals(2, calculator.subtract(5, 3));
    }
}
```

---

4. Run Tests and Generate Coverage Report 🚀

- **Step 1: Run Tests with JaCoCo**

Run the following command to generate code coverage:

```
mvn clean test jacoco:report
```

📌 This will generate the **JaCoCo report** at:
📂 `target/site/jacoco/index.html`

🔹 **Step 2: Analyze Code Coverage with SonarQube**

Now, run SonarQube analysis:

```
mvn sonar:sonar
```

👉 Go to **http://localhost:9000** and log in to **see the coverage report**! 🎯

---

5. Automating Code Coverage in CI/CD ⚙️

GitHub Actions (Maven + SonarQube + JaCoCo)

To automate code coverage in a CI/CD pipeline, create `.github/workflows/sonarqube.yml`:

```
Unset
name: Java CI with SonarQube

on:
  push:
    branches:
      - main

jobs:
  sonar:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Code
        uses: actions/checkout@v3

      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
          distribution: 'temurin'
          java-version: '17'
 - name: Build and Test with JaCoCo
        run: |
```

```
          mvn clean test jacoco:report

      - name: Run SonarQube Analysis
        run: |
          mvn sonar:sonar -Dsonar.host.url=http://localhost:9000
-Dsonar.login=admin
```

---

6. SonarQube Coverage Report Example 📊

Once the analysis is complete, SonarQube provides detailed **code coverage reports**:

✔ **Overall Code Coverage (%)**
✔ **Lines Covered vs. Uncovered**
✔ **Branch Coverage** (if all conditions are tested)
✔ **Duplicated Code Detection**
✔ **Code Smells & Vulnerabilities**

---

7. Best Practices for Code Coverage ✅

✔ **Aim for 80-90% coverage** (but don't chase 100%).
✔ Focus on **business-critical logic** (avoid testing trivial code).
✔ Ensure **branch and condition coverage** (not just line coverage).
✔ **Mock external dependencies** in unit tests (use Mockito).
✔ Automate **SonarQube analysis in CI/CD** (GitHub Actions, Jenkins).

Best Coding Practices & Industry Standards in Java

8. Summary

🔹 Step	✅ Action
Setup SonarQube & JaCoCo	Install SonarQube, add JaCoCo to Maven/Gradle
Write JUnit Tests	Use **JUnit 5** for unit testing
Run Tests & Generate Coverage	`mvn clean test jacoco:report`
Analyze with SonarQube	`mvn sonar:sonar`
Automate in CI/CD	Use **GitHub Actions / Jenkins**