Coding Standard and Guidelines of Java

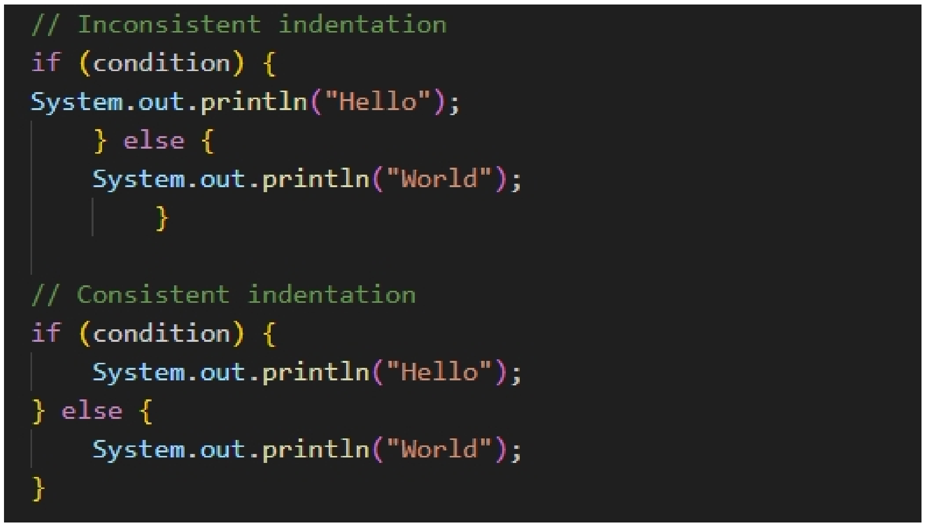
1 Basic Rules to Follow

- Use descriptive and appropriate names for all identifiers (variables, method names, class names, constants, etc.).
- Comment every 3-7 lines of code.
- Be neat.

2 Consistency

- Use consistent and clear indentation for better code readability.
- Follow a coding style guide, such as the Java Code Conventions or a team-agreed style guide.

Example:



```java
// Inconsistent indentation
if (condition) {
System.out.println("Hello");
    } else {
    System.out.println("World");
        }

// Consistent indentation
if (condition) {
    System.out.println("Hello");
} else {
    System.out.println("World");
}
```

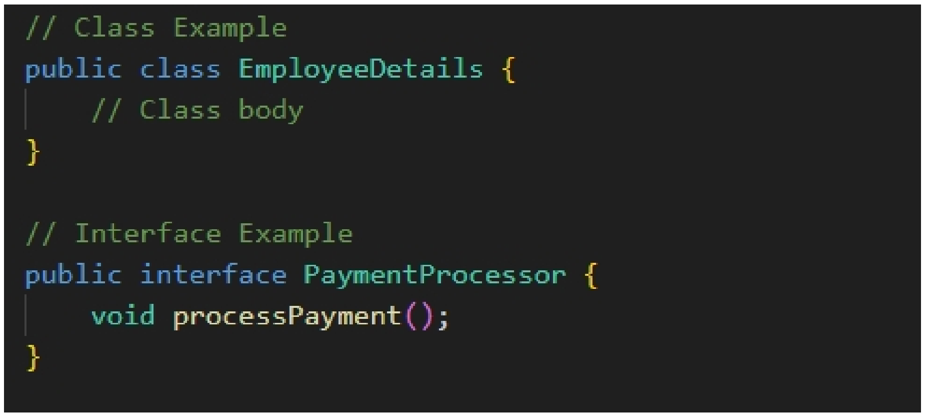
3 Naming Conventions

These are the rules for naming variables, methods, constants, classes, interfaces, etc. Generally, Java follows the camel case convention. It describes the following:

3.1 Class and Interface:

- The class and interface name must be noun, and the first letter of each internal word should be capitalized.

Example:

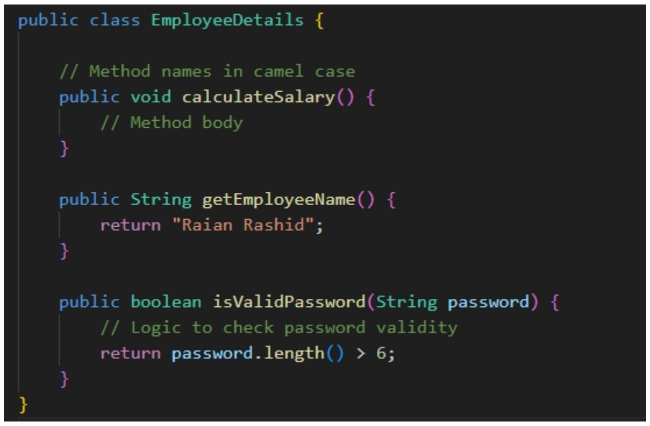


```java
// Class Example
public class EmployeeDetails {
    // Class body
}

// Interface Example
public interface PaymentProcessor {
    void processPayment();
}
```

3.2 Method Naming:

- The method name must be verb in mixed case, each first letter should be in lower case with the first letter of each internal word should be capitalized.

Example:

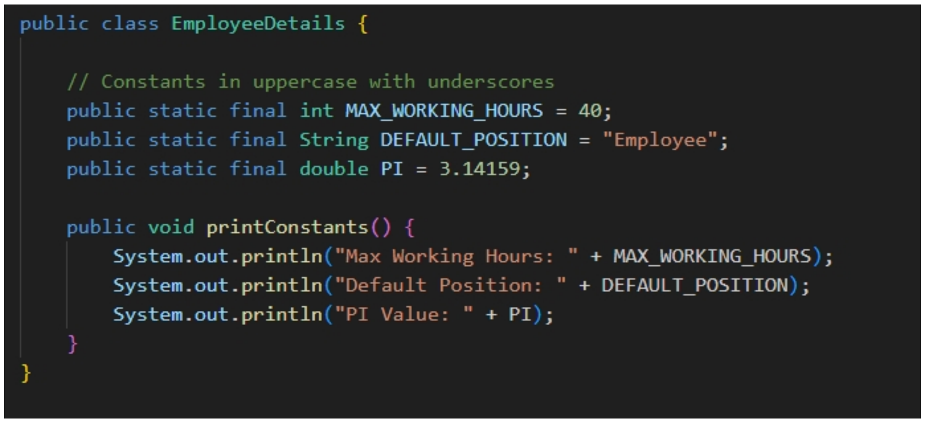


```java
public class EmployeeDetails {

    // Method names in camel case
    public void calculateSalary() {
        // Method body
    }

    public String getEmployeeName() {
        return "Raian Rashid";
    }

    public boolean isValidPassword(String password) {
        // Logic to check password validity
        return password.length() > 6;
    }
}
```

3.3 Constant Naming:

- All the constants should be in capital letters.

Example:

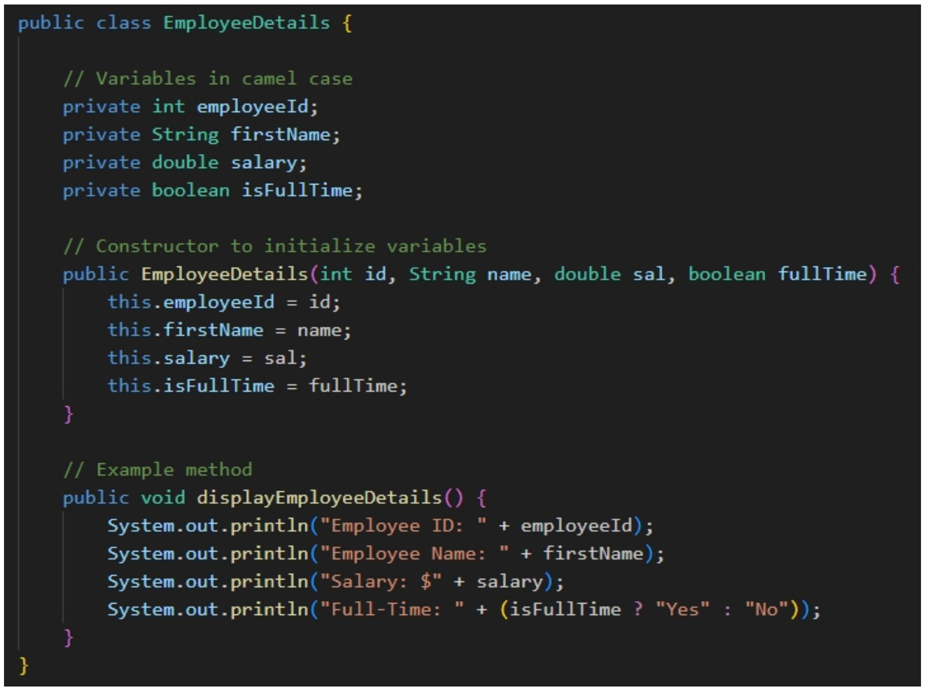


```java
public class EmployeeDetails {

    // Constants in uppercase with underscores
    public static final int MAX_WORKING_HOURS = 40;
    public static final String DEFAULT_POSITION = "Employee";
    public static final double PI = 3.14159;

    public void printConstants() {
        System.out.println("Max Working Hours: " + MAX_WORKING_HOURS);
        System.out.println("Default Position: " + DEFAULT_POSITION);
        System.out.println("PI Value: " + PI);
    }
}
```

3.4 Variable Naming:

- The variable name must be a meaningful letter or word.

Example:

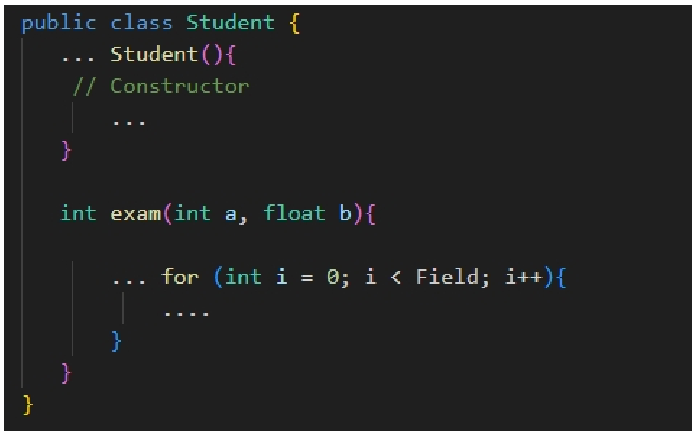


```java
public class EmployeeDetails {

    // Variables in camel case
    private int employeeId;
    private String firstName;
    private double salary;
    private boolean isFullTime;

    // Constructor to initialize variables
    public EmployeeDetails(int id, String name, double sal, boolean fullTime) {
        this.employeeId = id;
        this.firstName = name;
        this.salary = sal;
        this.isFullTime = fullTime;
    }

    // Example method
    public void displayEmployeeDetails() {
        System.out.println("Employee ID: " + employeeId);
        System.out.println("Employee Name: " + firstName);
        System.out.println("Salary: $" + salary);
        System.out.println("Full-Time: " + (isFullTime ? "Yes" : "No"));
    }
}
```

4 Curly Braces:

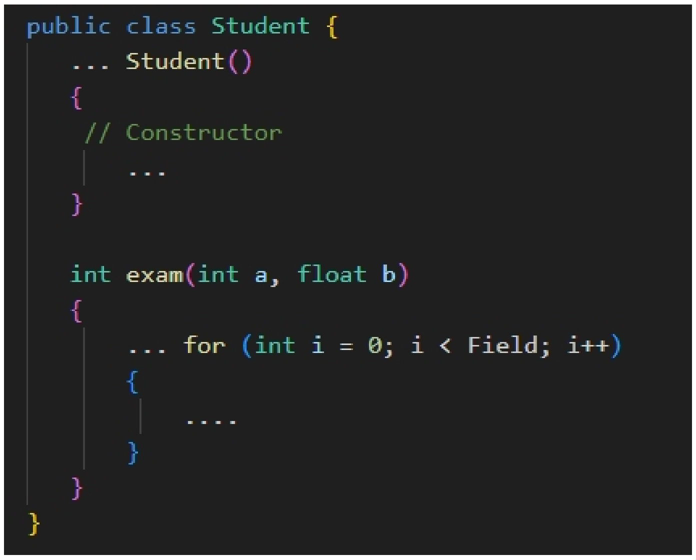
- No blank lines should be present after the opening brace or before the closing brace.
- A curly brace is applied at the end of the line that starts the class, method, loop, etc., and the closing brace is on a line by itself, lined up vertically with the start of the first line.

Example:



```
public class Student {
    ... Student(){
     // Constructor
        ...
    }

    int exam(int a, float b){

        ... for (int i = 0; i < Field; i++){
            ....
        }
    }
}
```

- Each curly brace is added on a new line, and the pair is aligned vertically. The preceding code snippet in this format would be as follows:



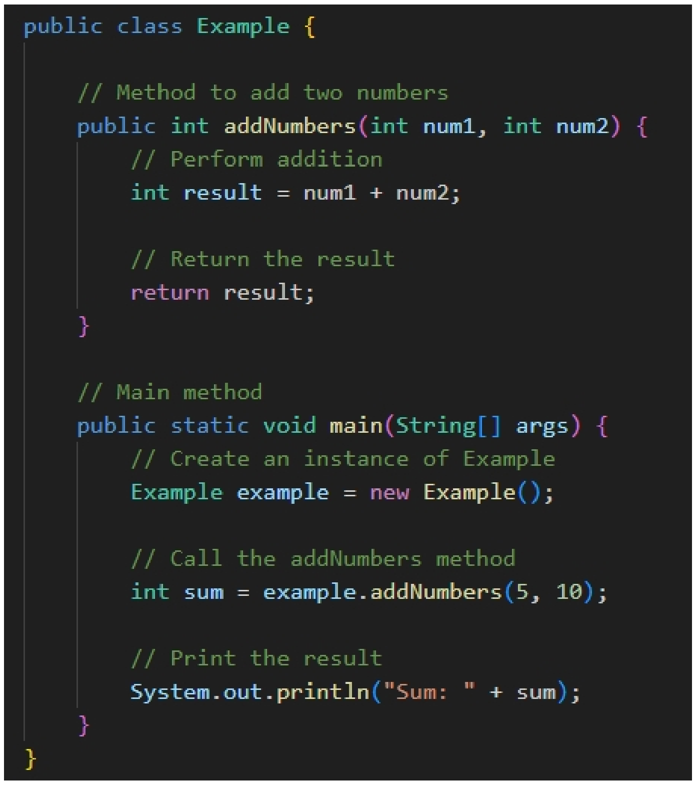
```
public class Student {
    ... Student()
    {
     // Constructor
        ...
    }

    int exam(int a, float b)
    {
        ... for (int i = 0; i < Field; i++)
        {
            ....
        }
    }
}
```

5 Indentation:

The indentation should be 4 spaces. By pressing the Tab key, we get exactly 8-spaces. Indentation can be achieved by the space character and the tab characters. The recognized standard for increasing readability of each line is:

- Apply indentation to alike items in a vertical list (such as end-of-line comments, and identifiers in declarations).
- Surround the binary operators (including assignment) by spaces.
- Follow a semicolon or comma by a space.
- Keep space between two keywords like "if", "while", "return", "catch", "switch", "for" and a succeeding parenthesis.
- Excess parentheses help to highlight the structure of expressions. But we should avoid many nested parentheses because it may lead to error.
- It is good to put an extra line to differentiate between the important piece of code.

Example:

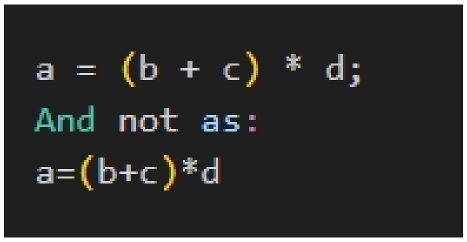


```java
public class Example {

    // Method to add two numbers
    public int addNumbers(int num1, int num2) {
        // Perform addition
        int result = num1 + num2;

        // Return the result
        return result;
    }

    // Main method
    public static void main(String[] args) {
        // Create an instance of Example
        Example example = new Example();

        // Call the addNumbers method
        int sum = example.addNumbers(5, 10);

        // Print the result
        System.out.println("Sum: " + sum);
    }
}
```

6 White Spaces:

- It also plays a vital role in readability of the program. Any mathematical operations (+, -, *, %, etc.) should be surrounded by a space character.

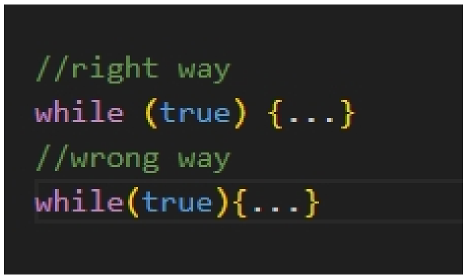
Example:



```
a = (b + c) * d;
And not as:
a=(b+c)*d
```

- Keywords that reserved in Java must followed by white spaces.

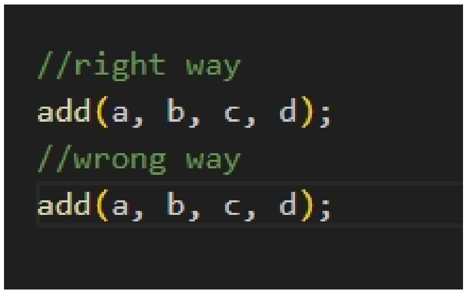
Example:



```
//right way
while (true) {...}
//wrong way
while(true){...}
```

- Always, put a space just after comma. It means comma must be followed by a white space.

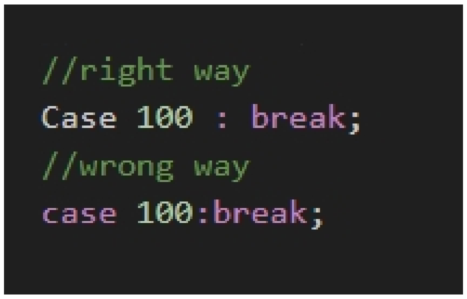
Example:



```
//right way
add(a, b, c, d);
//wrong way
add(a, b, c, d);
```

- Colons should be surrounded by white space.

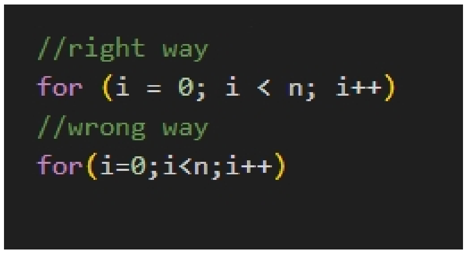
Example:



```
//right way
Case 100 : break;
//wrong way
case 100:break;
```

- Semicolons in for statements should be followed by a space character.

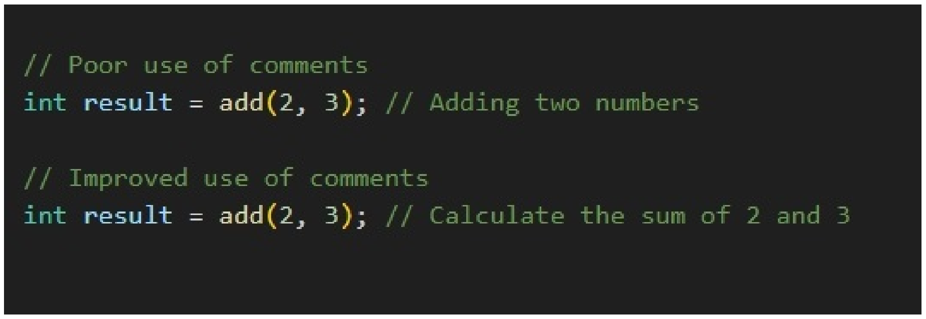
Example:



```
//right way
for (i = 0; i < n; i++)
//wrong way
for(i=0;i<n;i++)
```

7 Comments:

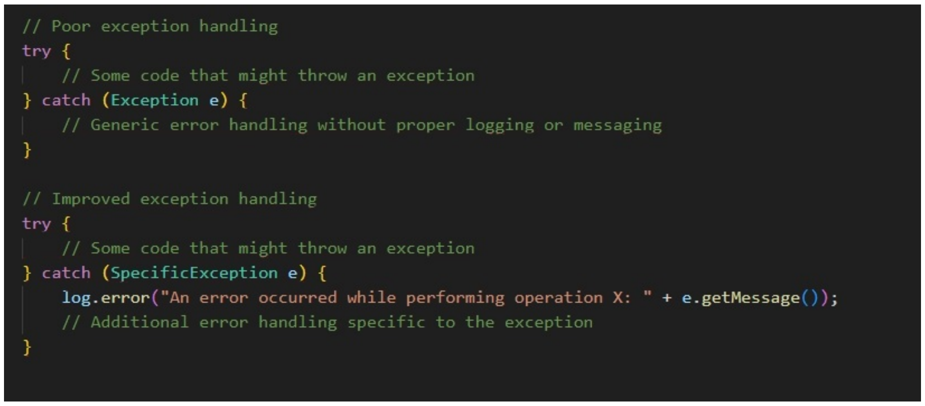
Comments play a crucial role in conveying the intent and logic behind your code. However, over-commenting or leaving outdated comments can clutter the code base and confuse readers. Use comments to explain complex algorithms, highlight important details, or clarify any potential pitfalls.
Example:



```
// Poor use of comments
int result = add(2, 3); // Adding two numbers

// Improved use of comments
int result = add(2, 3); // Calculate the sum of 2 and 3
```

8 Error Handling and Exceptions:

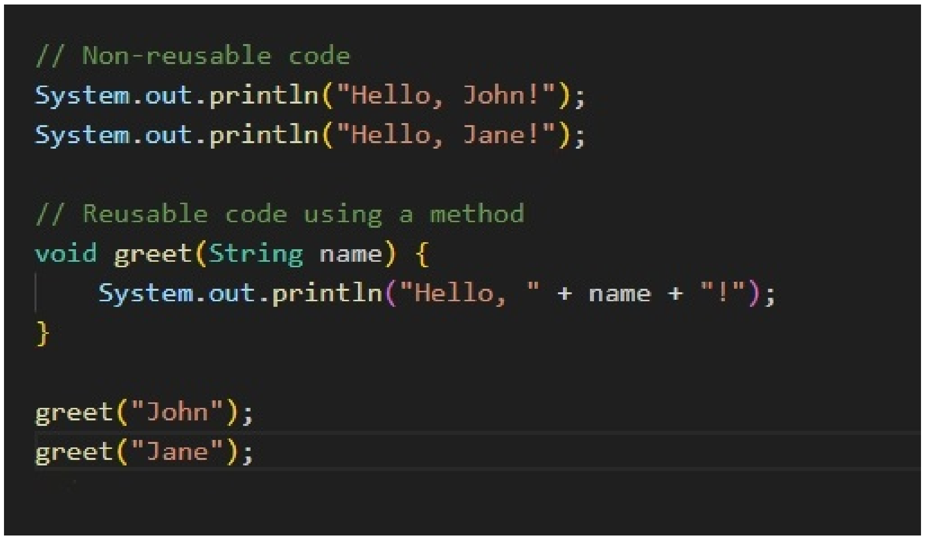
Java provides a robust exception handling mechanism, and utilizing it effectively can make your code more reliable. Always handle exceptions gracefully, using try-catch blocks, and provide meaningful error messages.
Example:



```java
// Poor exception handling
try {
    // Some code that might throw an exception
} catch (Exception e) {
    // Generic error handling without proper logging or messaging
}

// Improved exception handling
try {
    // Some code that might throw an exception
} catch (SpecificException e) {
    log.error("An error occurred while performing operation X: " + e.getMessage());
    // Additional error handling specific to the exception
}
```

9 Code Re-usability with Methods:

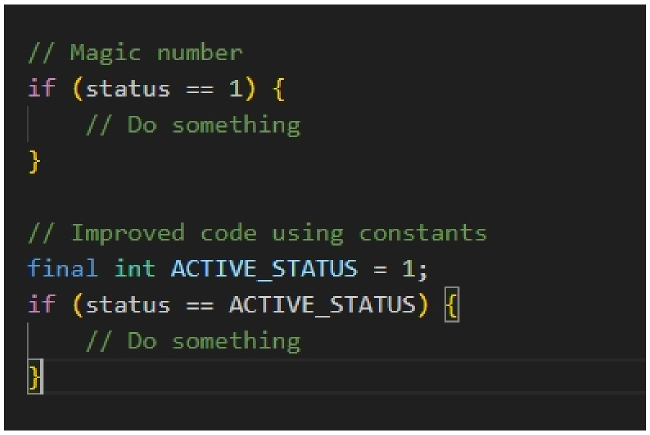
Encapsulating reusable code into methods not only promotes code re-usability but also enhances readability and maintainability. Break down complex tasks into smaller, self-contained methods. Example:



```java
// Non-reusable code
System.out.println("Hello, John!");
System.out.println("Hello, Jane!");

// Reusable code using a method
void greet(String name) {
    System.out.println("Hello, " + name + "!");
}

greet("John");
greet("Jane");
```

10 Magic Numbers:

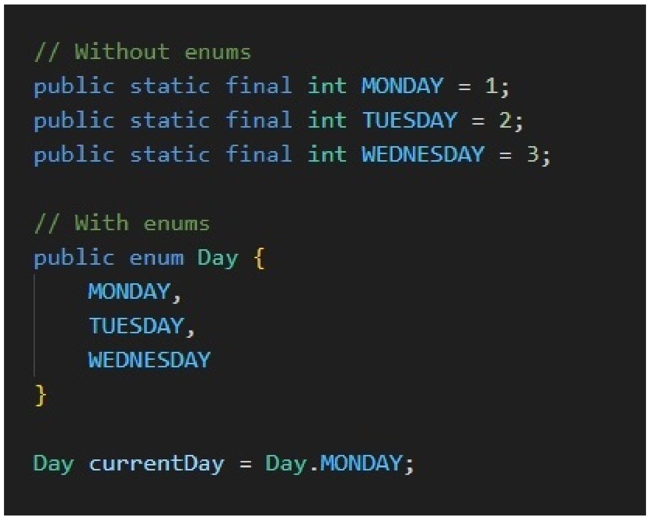
Magic numbers refer to hard-coded numerical values scattered throughout your code without any explanation. They make your code difficult to understand and maintain. Instead, define constants or variables with descriptive names to make the code more readable and maintainable. Example:



```java
// Magic number
if (status == 1) {
    // Do something
}

// Improved code using constants
final int ACTIVE_STATUS = 1;
if (status == ACTIVE_STATUS) {
    // Do something
}
```

11 Enumerations for Constants:

When you have a set of related constants, consider using Java enums. Enums provide a type-safe way to define a fixed number of values, improving code clarity and eliminating the use of magic numbers or string literals. Example:



```java
// Without enums
public static final int MONDAY = 1;
public static final int TUESDAY = 2;
public static final int WEDNESDAY = 3;

// With enums
public enum Day {
    MONDAY,
    TUESDAY,
    WEDNESDAY
}

Day currentDay = Day.MONDAY;
```

References:

1. https://medium.com/@raghunathchavva/java-coding-standards-and-best-practices-write-high-quality-code-fafc4ad070c3
2. https://www.javatpoint.com/coding-guidelines-in-java
3. https://www.geeksforgeeks.org/comments-in-java/?ref=next$_{a}rticle$