EXP53-J. Use parentheses for precedence of operation

Programmers frequently make errors regarding the precedence of operators because of the unintuitively low precedence levels of `&`, `|`, `^`, `<<`, and `>>`. Avoid mistakes regarding precedence through the suitable use of parentheses, which also improves code readability. The precedence of operations by the order of the subclauses is defined in the Java Tutorials [Tutorials 2013].

Although it advises against depending on parentheses for specifying evaluation order EXP05-J. Do not follow a write by a subsequent write or read of the same object within an expression applies only to expressions that contain side effects.

Noncompliant Code Example

The intent of the expression in this noncompliant code example is to add the variable `OFFSET` to the result of the bitwise logical AND between `x` and `MASK`:

```
public static final int MASK = 1337;
public static final int OFFSET = -1337;

public static int computeCode(int x) {
  return x & MASK + OFFSET;
}
```

According to the operator precedence guidelines, the expression is parsed as the following:

```
x & (MASK + OFFSET)
```

This expression is evaluated as follows, resulting in the value 0:

```
x & (1337 - 1337)
```

Compliant Solution

This compliant solution uses parentheses to ensure that the expression is evaluated as intended:

```
public static final int MASK = 1337;
public static final int OFFSET = -1337;

public static int computeCode(int x) {
  return (x & MASK) + OFFSET;
}
```

Noncompliant Code Example

In this noncompliant code example, the intent is to append either "0" or "1" to the string "`value=`":

```
public class PrintValue {
  public static void main(String[] args) {
    String s = null;
    // Prints "1"
    System.out.println("value=" + s == null ? 0 : 1);
  }
}
```

However, the precedence rules result in the expression to be printed being parsed as `("value=" + s) == null ? 0 : 1`.

Compliant Solution

This compliant solution uses parentheses to ensure that the expression evaluates as intended:

```
public class PrintValue {
  public static void main(String[] args) {
    String s = null;
    // Prints "value=0" as expected
    System.out.println("value=" + (s == null ? 0 : 1));
  }
}
```

Applicability

Mistakes regarding precedence guidelines can cause an expression to be evaluated in an unintended way, which can lead to unexpected and abnormal program behavior.

Parentheses may be omitted from mathematical expressions that follow the algebraic precedence rules. For instance, consider the following expression:

```
x + y * z
```

By mathematical convention, multiplication is performed before addition; parentheses are redundant in this case:

```
x + (y * z)
```

Detection of all expressions using low-precedence operators without parentheses is straightforward. Determining the correctness of such uses is infeasible in the general case, although heuristic warnings could be useful.

Automated Detection

Tool	Version	Checker	Description
CodeSonar	5.2p0	**FB.CORRECTNESS.BSHIFT_WRONG_ADD_PRIORITY**	Possible bad parsing of shift operation
SonarQube	6.7	**S864**	

Bibliography

[ESA 2005]	Rule 65, Use parentheses to explicitly indicate the order of execution of numerical operators
[Tutorials 2013]	Expressions, Statements, and Blocks