2IP90 Java Coding Standard

This coding standard is a simplified version of the *Google Java Style Guide*. It lists the programming style rules for writing Java programs in our programming course. With this coding standard, we aim at easily readable and modifiable Java source code. Adhere to this coding standard to help human readers of your code, including your assignment partner, the teachers, and *yourself*, to understand your code more easily. Furthermore, it prevents errors. When you do make an error, it helps you to locate and fix them more quickly. Finally, don't lose points on your assignments or during the exam by ignoring this coding standard.

Naming conventions

1. Use meaningful names and spell them correctly. Keep names short, but not at the cost of comprehensibility. Don't use single character names, except for loop counters, like `i`, `j`, and `k`, or concepts commonly referred to by a single character, like `x` for an x coordinate.

2. Class and record names should be a noun and start with an uppercase letter. If the name consists of multiple words, add words without a separator. Each word starts with an uppercase letter.

   We call this way of writing names *CamelCase*.

   ```
   // Bad
   class CalculateAge { /* ... */ }
   class Age_Calculator { /* ... */ }
   record point(int x, int y) { /* ... */ }

   // Good
   class AgeCalculator { /* ... */ }
   record Point(int x, int y) { /* ... */ }
   ```

3. Start variable names, parameter names, and method names with a lowercase letter. If the name consists of multiple words, use *camelCase*. Method names should be a verb.

   ```
   // Bad
   int DayOfBirth = 23;
   int month_of_birth = 5;
   int year_Of_Birth = 2001;

   void calculate_Age() { /* ... */ }
   void ageCalculator(Date DOB) { /* ... */ }

   // Good
   int dayOfBirth = 23;
   int monthOfBirth = 5;
   int yearOfBirth = 2001;

   void calculateAge(Date dateOfBirth) { /* ... */ }
   ```

4. Exception to rule 2 and 3: Write constant names in all uppercase, separate words with an underscore ("_").

```
// Bad
static final int yearInDays = 365;
static final int nr_of_months = 12;

// Good
static final int YEAR_IN_DAYS = 365;
static final int NR_OF_MONTHS = 12;
```

Indentation

5. Indent consistently with a fixed multiple of four spaces.

6. Don't use tabs. Configure your editor to change a tab to four spaces.

Line length

7. Don't put more than 100 characters on one line.

Put one item per line

8. Write variable declarations and constant definitions on separate lines.

```
// Bad
static final WIDTH = 800, HEIGHT = 600;
int x, y;

// Good
static final WIDTH = 800;
static final HEIGHT = 600;

int x;
int y;
```

9. Write statements and assignments on separate lines.

```
// Bad
x = e.getX(); y = e.getY(); if (x >= WIDTH) { x = WIDTH - 1; }

// Good
x = e.getX();
y = e.getY();

if (x >= WIDTH) {
    x = WIDTH - 1;
}
```

White space

10. Write an empty line before *and* after:
    - A block of constant definitions, instance variable declarations, or local variable declarations
    - A class, method or record declaration
11. Never put a space before a comma (","), colon (":"), or semicolon (";"). Always put a space after them.
12. Put a space before *and* after:
    - keywords, e.g., `class`, `if`, `void`, `else`, `try`, `catch`, `new`, etc.
    - operators, e.g., =, ==, *, %, -=, &&, <=, etc., except for -- and ++.
13. Don't put a space after an opening bracket ("(", "[", "<" or "{") or before a closing bracket (")", "]", ">", or "}").

Examples

```
// Bad
int[ ] a = { 1, 2, 3, 4 };
if( x<= 4 ){ x --;
}else{
 x=x*5;
}

// Good
int[] a = {1, 2, 3, 4};

if (x <= 4) {
    x--;
} else {
    x = x * 5;
}
```

Curly brackets

14. Use curly brackets, or braces, after the following statements:
    - `if`, `switch`, and `else`, unless this `else` is immediately followed by `if` for a multiway selection.
    - `while`, `for`, and `do`
    - `try`, `catch`, and `finally`
15. Put opening braces on the same line as the statement they belong to.
16. Put the closing braces on a new line. Put keywords like `else`, `catch`, and `finally` on the same line as these closing braces.

Examples

```
// Bad
if (a.length < 10)
{
while (x < a.length) x = x + 1;}
else if (a.length < 100)
{ x = x * 3; } else x = 0;

// Good
if (a.length < 10) {
    while (x < a.length) {
        x = x + 1;
    }
} else if (a.length < 100) {
    x = x * 3;
} else {
    x = 0;
}
```

Comments

17. Explain each variable and constant with a comment unless they are self-explanatory.
18. Document non-private classes, methods, and records with a JavaDoc comment, except for the `main` method and constructors.

Examples

```
// Bad
final static int W = 800;
final static int H = 600;

void drawRectangle(int x, int y) { /* ... */ }

// Good
final static int WIDTH = 800; // The drawing canvas width in pixels.
final static int CANVAS_HEIGHT_IN_PX = 600;

/**
 * Draw a rectangle at position (x, y)
 *
 * @param x  x coordinate
 * @param y  y coordinate
 */
void drawRectangle(int x, int y) { /* ... */ }
```