Good Programming Style

Good programming style

The goal of good style is to make your code more readable.

By you and by others.

Rule #1: use good (meaningful) names

```
String a1;
int a2;
double b;              // BAD!!

String firstName;      // GOOD
String lastName;       // GOOD
int temperature;       // GOOD
```

Rule #2: Use indentation

```
public static void main (String[] arguments) {
    int x = 5;
    x = x * x;
    if (x > 20) {
        System.out.println(x + " is greater than 20.");
    }
    double y = 3.4;
}
```

Have a demo with no indentation

Ctrl-shift-F to auto-format the file

Rule #3: Use whitespaces

Put whitespaces in complex expressions:

```
// BAD!!
double cel=fahr*42.0/(13.0-7.0);


// GOOD
double cel = fahr * 42.0 / (13.0 - 7.0);
```

Rule #3: Use whitespaces

Put blank lines to improve readability:

```
public static void main (String[] arguments) {

     int x = 5;
     x = x * x;

     if (x > 20) {
          System.out.println(x + " is > 20.");
     }

     double y = 3.4;
}
```

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
          ...
} else if (hours > 60) {
          ...
} else if (basePay >= 8.0 && hours <= 60) {
           ...
}
```

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
              ...
} else if (hours > 60) {
              ...
} else if (basePay >= 8.0 && hours <= 60){
              ...
}
```

BAD

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
             ...
} else if (hours > 60) {
             ...
} else {
             ...
}
```

Good programming style (summary)

Use good names for variables and methods

Use indentation

Add whitespaces

Don't duplicate tests

What is a good program?

Correct / no errors

Easy to understand

Easy to modify / extend

Good performance (speed)

Consistency

Writing code in a consistent way makes it easier to write and understand

Programming “style” guides: define rules about how to do things

Java has some widely accepted “standard” style guidelines

Naming

Variables: Nouns, lowercase first letter, capitals separating words

x, shape, highScore, fileName

Methods: Verbs, lowercase first letter

getSize(), draw(), drawWithColor()

Classes: Nouns, uppercase first letter

Shape, WebPage, EmailAddress

Debugging

The process of finding and correcting an error in a program

A fundamental skill in programming

Step 1: Don’t Make Mistakes

Don’t introduce errors in the first place

- Reuse: find existing code that does what you want
- Design: think before you code
- Best Practices: Recommended procedures/techniques to avoid common problems

Step 2: Find Mistakes Early

Easier to fix errors the earlier you find them

- Test your design
- Tools: detect potential errors
- Test your implementation
- Check your work: assertions

Tools: Eclipse Warnings

Warnings: may not be a mistake, but it likely is.

Suggestion: always fix all warnings

Extra checks: FindBugs and related tools

Unit testing: JUnit makes testing easier

Step 3: Reproduce the Error

- Figure out how to repeat the error
- Create a minimal test case

Go back to a working version, and introduce changes one at a time until the error comes back

Eliminate extra stuff that isn’t used

Step 4: Generate Hypothesis

What is going wrong?

What might be causing the error?

Question your assumptions: “x can’t be possible:” What if it is, due to something else?

Step 5: Collect Information

If x is the problem, how can you verify? Need information about what is going on inside the program

System.out.println() is very powerful

Eclipse debugger can help

Step 6: Examine Data

Examine your data

Is your hypothesis correct?

Fix the error, or generate a new hypothesis