Beginning Style
Be consistent!

- Most times, you will enter an ongoing project, with established style rules
  - Follow them even if you don’t like them
- In this course you will be working in teams with various other people
  - We’ll all use the same set of style rules
Do it right the first time

- You only write code once, but you read it many times while you’re trying to get it to work
  - Good style makes it more readable and helps you get it right!
- You’re working on a large project, so you use good style...
  - ...but you need a tool to help you do one little job, so you slap it together quickly
  - Guess which program will be around longer and used by more people?
Always indent statements that are nested inside (under the control of) another statement

```java
if (itemCost <= bankBalance) {
    writeCheck(itemCost);
    bankBalance = bankBalance - itemCost;
}
```

The open brace always goes at the end of a line

The matching close brace lines up with the statement being closed

Indentation should be consistent throughout the program

For Java, 4 spaces is the standard
Break up long lines

- Keep your lines short enough to be viewed and printed
- Many people use 72 or 80 character limits
- Suggestions on where to break a long line:
  - It’s *illegal* to break a line within a quoted string
  - Break after, not before, operators
  - Line up parameters to a method
  - *Don’t* indent the second line of a control statement with a long test so that it lines up with the statements being controlled
Don’t use “hard” tabs

- A **hard tab** is an actual *tab character* in your text
  - It tells the program to go to the next *tab stop* (wherever that is)
  - Not every program puts tab stops in the same place
- If you use hard tabs to indent, sooner or later your nice indentation will be ruined
- Good editors can be set to use **soft tabs** (your tab characters are replaced with spaces)
  - When you hit the tab key, the editor puts spaces into your file, not tab characters
  - With soft tabs, your indentation is always safe
  - The default Eclipse indentation, with mixed tabs and spaces, is *wrong*
Using spaces

- Use spaces around all binary operators except “dot”:
  
  ```
  if (n > 1 && n % 2 == 1) n = 3 * n + 1;
  ```

- Do not use spaces just within parentheses:
  
  ```
  if (x < 0) x = -x; // don’t do this
  ```

- Use a space before and after the parenthesized test in a control statement:
  
  ```
  if (x < 0) {...}
  while (x < 0) {...}
  ```

- Do not use a space between a method name and its parameters; do put a space after each comma:
  
  ```
  int add(int x, int y) {...}
  a = add(3, k);
  ```
Use meaningful names

- Names should be chosen very carefully, to indicate the purpose of a variable or method
  - If the purpose changes, the name should be changed
  - *Spend a little time to choose the best name for each of your variables and methods!*

- Long, multiword names are common in Java
  - However, if a name is too long, maybe you’re trying to use it for too many purposes
    - Don’t change the name, separate the purposes

- Don’t abbreviate names
  - Let Eclipse help you—start typing the name, then hit control-space
  - Very common abbreviations, such as *max* for “maximum”, are OK
Meaningful names: exceptions I

- It is common practice to use \( i \) as the index of a for-loop, \( j \) as the index of an inner loop, and \( k \) as the index of a third-level loop.
- This is almost always better than trying to come up with a meaningful name.
- Example:
  ```java
  for (int i = 1; i <= 10; i++) {
    for (int j = 1; j <= 10; j++) {
      System.out.println("   " + (i * j));
    }
  }
  ```
Meaningful names: exceptions II

- Local variables in methods may be given short, simple names, **if:**
  - The purpose of the variable is obvious from context, *and*
  - The variable is used only briefly, in a small part of the program
- But *never* use meaningless names for fields (class or instance variables) or classes or methods
If variables have no special meaning, you can use names that reflect their types

For example, if you are writing a general method to work with *any* strings, you might name them `string1`, `string2`, etc.

Alternatively, you can use very short names

- `s`, `t`, `u`, or `s1`, `s2`, etc. are often used for Strings
- `p`, `q`, `r`, `s` are often used for booleans
- `w`, `x`, `y`, `z` are often used for real numbers
Naming classes and interfaces

- Capitalize the first letter of each word, including the first word:
  
  `PrintStream`, `Person`, `ExemptEmployee`

- Use nouns to name classes:
  
  `ExemptEmployee`, `CustomerAccount`
  
  - Classes are supposed to represent *things*

- Use adjectives to name interfaces:
  
  `Comparable`, `Printable`
  
  - Interfaces are supposed to represent *features*
Naming variables

- Capitalize the first letter of each word except the first: total, maxValue

- Use nouns to name variables: balance, outputLine
  - Variables are supposed to represent values
Naming constants

- A constant is an identifier whose value, once given, cannot be changed
- Constants are written with the keyword `final`, for example:
  - `final int FIVE = 5;`
  - `final float AVOGADROS_NUMBER = 6.022E23;`
- Constants are written in ALL_CAPITALS, with underscores between words
Naming methods

- Capitalize the first letter of each word except the first: display, displayImage
  - Methods are capitalized the same as variables

- Use verbs when naming methods: displayImage, computeBalance
  - Methods are supposed to do something
Keep your methods short

- Methods give you a chance to *name* what you are doing
  - Well-chosen names can greatly improve readability
  - If your method does A, then B, then C, it will probably improve readability to make A, B, and C into methods

- Eclipse makes it easy to *refactor* a long method
  - Refactoring is changing the structure of a program, without changing in any way what the program does
  - In Eclipse,
    - Choose a range of lines
    - Choose `Refactor -> Extract Method`
    - Give your new method a name, and Eclipse does the rest
  - This refactoring is possible if (and only if) the resultant method needs to return only a single value
Correct style made easy

- In Eclipse,
  - Go to Window → Preferences → Java → Code Style → Formatter
  - Under Select a profile: choose Java conventions [built-in] and click Edit...
  - In the Indentation tab, set Tab policy: to Spaces only, and set both Indentation size: and Tab size: to 4
  - Enter a new Profile name:, for example, Java Conventions [corrected]
  - Use these conventions henceforth

- Select some or all of your code and choose Source → Format

- To simply indent correctly, without reformatting, select some lines and choose Source → Correct Indentation or just type ctrl-I.
“Where a calculator on the ENIAC is equipped with 18,000 vacuum tubes and weighs 30 tons, computers of the future may have only 1,000 vacuum tubes and perhaps weigh 1½ tons.”

—Popular Mechanics, March 1949