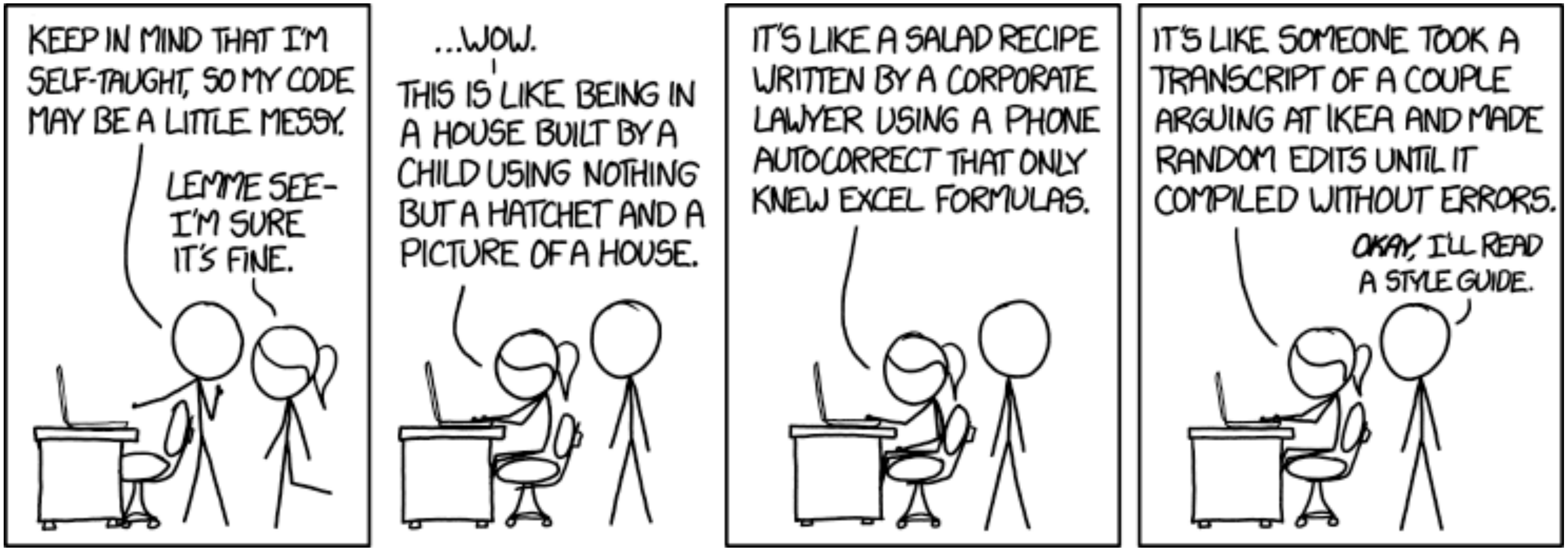

KEEP IN MIND THAT I'M SELF-TAUGHT, SO MY CODE MAY BE A LITTLE MESSY.
LEMME SEE– I'M SURE IT'S FINE.
...WOW.
THIS IS LIKE BEING IN A HOUSE BUILT BY A CHILD USING NOTHING BUT A HATCHET AND A PICTURE OF A HOUSE.
IT'S LIKE A SALAD RECIPE WRITTEN BY A CORPORATE LAWYER USING A PHONE AUTOCORRECT THAT ONLY KNEW EXCEL FORMULAS.
IT'S LIKE SOMEONE TOOK A TRANSCRIPT OF A COUPLE ARGUING AT IKEA AND MADE RANDOM EDITS UNTIL IT COMPILED WITHOUT ERRORS.
OKAY, I'LL READ A STYLE GUIDE.

Title text: I honestly didn't think you could even USE emoji in variable names. Or that there were so many different crying ones.

Good Code II

CSCI 2134: Software Development

Agenda

- Lecture Contents
    - Brightspace Quiz
    - Writing Readable code leads to understandable code
- Readings:
    - This Lecture: Chapter 20, 31
    - Next Lecture: Chapter 21

Maximizing Readability and Understandability

- Code that is easy to comprehend at the source level is a precursor to nearly all other software quality characteristics

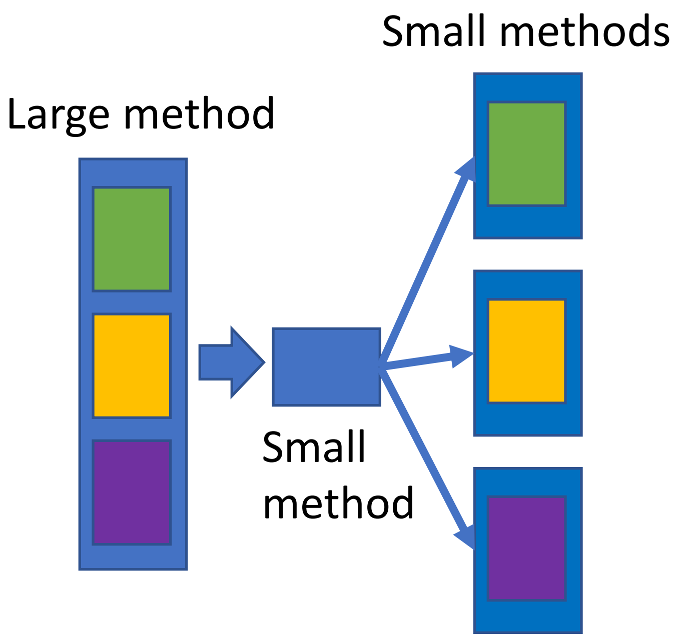
- Readability depends on
  - Logical structure of the code
  - Documentation
  - Consistent formatting, layout, and style

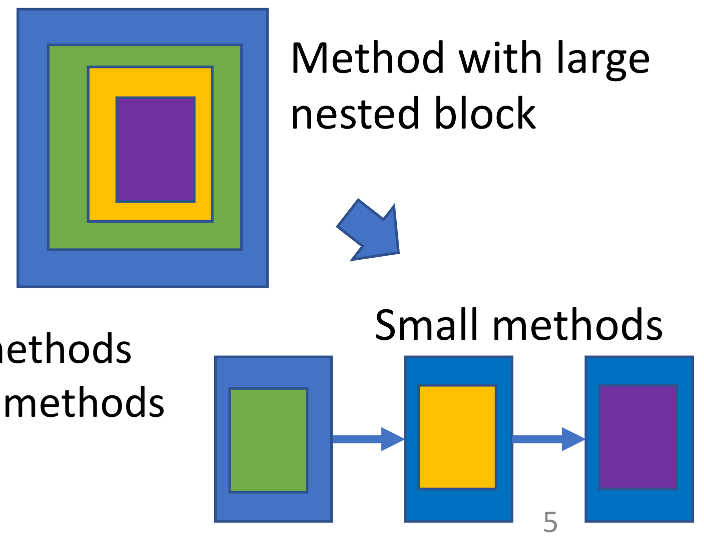
- Each of these is important and plays a role in creating readable code

- This is why part of your assignment marks in past courses went towards code readability

Logical Structure of Code

- Readable code is divided into small logical units
  - Packages are divided into files and classes
  - Classes are divided into nested classes, methods and attributes
  - Methods are divided into code blocks
- Files should contain no more than one public class
  - What Java requires
- Classes
  - Should have a single purpose (cohesion)
  - May contain as many methods, nested classes, and variables as needed
  - May span many lines
- Methods should (normally) fit on one screen
  - Large public methods can usually be transformed into a small public method that calls several small private methods
- Code blocks should (normally) fit on the screen
  - Large code blocks can often be transformed into smaller ones that call private methods
  - Deeply nested code can also be transformed into simpler code that calls private methods





Documentation

Documentation describes:

- Authorship of the code, who wrote it and when
- Purpose of the code
- Assumptions not self-evident in the code
- Reasons for what the code is doing
- Sources and origins of the code (citations where needed)

```
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;            // start of longest seq.
    int curIndex = 0;         // start of current seq.
    int maxCount = 1;         // length of longest seq
    int count = 1;            // length of current seq
    int prev = in.nextInt();  // prev num in seq.
    int seq = 1;              // seq # of cur read num

    /* assume integers are separated by spaces
     */
    while (in.hasNextInt()) {
      int next = in.nextInt();

      /* Next number is either part of current sequence
       * or start of next
       */
      if (next > prev) {
       . . .
```

Documentation for Logical Units of Code

Logical Unit	Documentation Required
File	(1) Authorship; (2) purpose of code in this file; (3) general assumptions and information about the code; and (4) sources of origin if the code was not written by your organization
Class	(1) Purpose of class and what it represents in the system; (2) Any special assumptions about the class, e.g., this is where the program starts to run
Class variables	(1) Purpose of variables if not clear from the name
Class methods	(1) Purpose and description of the method; (2) assumptions specific to the method; (3) parameters and assumptions about the parameters; (4) what the method returns;
Code blocks	(1) Assumptions made by the code block; (2) Reason for the code block (if not clear)
Local variables	(1) Purpose of local variables if not clear from the name
Elsewhere	Anything you believe would make the code-reader's job easier

The Code Documents **What** It Is Doing

- Ever seen the following (on the right)?
- The comments on the right are not very helpful!

- Rule: If your comment is simply restating what your code is doing, it is unnecessary and makes your code less readable!

```
...
/* Variable declarations
 */
int size; // declare integer variable
String name; // declare String variable

...

/* Loop over the list
 */
for (String s : list) {
  ...
}
```

Layout and Style

- Good visual layout shows the logical structure of a program
- Consistency of layout and style is more important than having “the right” layout and style
- “Good” layout and style is often subjective. Separate the objective gains from your subjective preferences
- Example: **Question**: Which is better? A or B?
- **Answer:** Neither! Whichever your team or your boss wants you to use!

Style A	Style B
`if (a == b) {`	`if (a == b)`
`    …`	`{`
`}`	`    …`
	`}`

Objectives of Layout and Style

- Accurately represent the logical structure of the code
- Consistently represent the logical structure of the code
- Improve readability
- Withstand modification
  - When modifying the code, the layout and style should be easy to match and make the modifications easy to do.
  - A modification should not break the layout and style of the code

Problems with this code?

- No indentation
- Poor spacing
- No comments
- No identification
- Not readable!
- Not understandable! ☹

```
public class LongestIncreasingSubsequence {
static void main(String [] a) {
Scanner in = new Scanner(System.in);
int index = 0;
int curIndex = 0;
int maxCount = 1;
int count = 1;
int prev = in.nextInt();
int seq = 1;
while(in.hasNextInt()) {
int next = in.nextInt();
if(next > prev) {
count++;
if(count > maxCount) {
maxCount = count;
index = curIndex;
}
} else {
curIndex = seq;
count = 1
}
prev = next;
seq++;
}
System.out.println(index);
}
}
```

```
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);
    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;
    while (in.hasNextInt()) {
      int next = in.nextInt();
      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      prev = next;
      seq++;
    }
    System.out.println(index);
  }
}
```

```
public class LongestIncreasingSubsequence {
static void main(String [] a) {
Scanner in = new Scanner(System.in);
int index = 0;
int curIndex = 0;
int maxCount = 1;
int count = 1;
int prev = in.nextInt();
int seq = 1;
while(in.hasNextInt()) {
int next = in.nextInt();
if(next > prev) {
count++;
if(count > maxCount) {
maxCount = count;
index = curIndex;
}
} else {
curIndex = seq;
count = 1
}
prev = next;
seq++;
}
System.out.println(index);
}
}
```

A or B?

Use Indentation

- Proper indentation allows the reader to easily identify which code is part of which scope.
- Indentation gives code a visual structure that corresponds to the structure of the code itself.
- How much to indent?
  - Too much makes for long lines to connect
  - Too little makes it hard to see the nesting
- **Guideline**: 2 or 4 spaces is typically good
- **Do not mix spaces and tabs in your indenting**

```
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);
    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;
    while (in.hasNextInt()) {
      int next = in.nextInt();
      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      prev = next;
      seq++;
    }
    System.out.println(index);
  }
}
```

```java
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);
    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;
    while (in.hasNextInt()) {
      int next = in.nextInt();
      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      prev = next;
      seq++;
    }
    System.out.println(index);
  }
}
```

```java
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while(in.hasNextInt()) {
      int next = in.nextInt();

      if(next > prev) {
        count++;
        if(count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }

      prev = next;
      seq++;
    }

    System.out.println(index);
  }
}
```

A or B?

Use Blank Lines to Separate Blocks of Code

- Blank lines separate blocks of functionality in your program
- **Guideline**: Blocks of
  - Variable declarations
  - Decision statements
  - Loops
  - **Etc.**

  Should be separated by blank lines

```
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while(in.hasNextInt()) {
      int next = in.nextInt();

      if(next > prev) {
        count++;
        if(count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }

      prev = next;
      seq++;
    }

    System.out.println(index);
  }
}
```

```
/**
 * CSCI 1110
 * @author: Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while (in.hasNextInt()) {
      int next = in.nextInt();

      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      . . .
```

```
public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while(in.hasNextInt()) {
      int next = in.nextInt();

      if(next > prev) {
        count++;
        if(count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }

      prev = next;
      seq++;
    }

    System.out.println(index);
  }
}
```

A or B?

Each File Should Have an Id Block

- The ID block at the top helps identify your work and the purpose of the code in the file
- The ID block should include
  - Purpose / course
  - Author
  - Description of what the code does
  - Any additional information

```
/**
 * CSCI 1110
 * @author: Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while (in.hasNextInt()) {
      int next = in.nextInt();

      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      . . .
```

```java
/**
 * CSCI 1110
 * @author: Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;
    int curIndex = 0;
    int maxCount = 1;
    int count = 1;
    int prev = in.nextInt();
    int seq = 1;

    while (in.hasNextInt()) {
      int next = in.nextInt();

      if (next > prev) {
        count++;
        if (count > maxCount) {
          maxCount = count;
          index = curIndex;
        }
      } else {
        curIndex = seq;
        count = 1
      }
      . . .
```

```java
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;           // start of longest seq.
    int curIndex = 0;        // start of current seq.
    int maxCount = 1;        // length of longest seq
    int count = 1;           // length of current seq
    int prev = in.nextInt();  // prev num in seq.
    int seq = 1;             // seq # of cur read num

    /* Assume list is separated by spaces
     */
    while (in.hasNextInt()) {
      int next = in.nextInt();

      /* Next number may not be part of this sequence
       */
      if (next > prev) {
        . . .
```

A or B?

Comments in the Code

- Code comments describe the purpose and behaviour of parts of your code
- Comments should
  - Describe the purpose and reason for the code.
  - Explain complex algorithms
  - Be close to the code itself
- Comments are not needed for each line.

```
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;          // start of longest seq.
    int curIndex = 0;       // start of current seq.
    int maxCount = 1;       // length of longest seq
    int count = 1;          // length of current seq
    int prev = in.nextInt();  // prev num in seq.
    int seq = 1;            // seq # of cur read num

    /* Assume list is separated by spaces
     */
    while (in.hasNextInt()) {
      int next = in.nextInt();

      /* Next number may not be part of this sequence
       */
      if (next > prev) {
       . . .
```

```
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main  (String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;          // start of longest seq.
     int curIndex = 0;            // start of current seq.
    int maxCount = 1;       // length of longest seq
    int count = 1;            // length of current seq
     int prev = in.nextInt();  // prev num in seq.
    int seq = 1;            // seq # of cur read num

    /* Assume list is separated by spaces
     */
    while (in.hasNextInt() ) {
      int next =  in.nextInt();

      /* Next number may not be part of this sequence
       */
      if( next > prev) {
       . . .
```

A or B?

```
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;          // start of longest seq.
    int curIndex = 0;       // start of current seq.
    int maxCount = 1;       // length of longest seq
    int count = 1;          // length of current seq
    int prev = in.nextInt();  // prev num in seq.
    int seq = 1;            // seq # of cur read num

    /* Assume list is separated by spaces
     */
    while (in.hasNextInt()) {
      int next = in.nextInt();

      /* Next number may not be part of this sequence
       */
      if (next > prev) {
       . . .
```

Make Code Consistent

- Coding style should be consistent throughout the code.
- Things to look for:
  - Brace style : choose one and stick to it
  - Spacing in parentheses
  - Spacing between operators
  - Indentations
  - Etc.

```
/**
 * CSCI 1110
 * @author Alex Brodsky
 * @description: This program computes the start
 * of the longest increasing subsequence of a
 * sequence of numbers
 */

public class LongestIncreasingSubsequence {
  /* main method, where the program starts running
   * @params: Strings [] : command line parameters
   * @return: none
   */
  static void main(String [] a) {
    Scanner in = new Scanner(System.in);

    int index = 0;           // start of longest seq.
    int curIndex = 0;        // start of current seq.
    int maxCount = 1;        // length of longest seq
    int count = 1;           // length of current seq
    int prev = in.nextInt();  // prev num in seq.
    int seq = 1;             // seq # of cur read num

    /* Assume list is separated by spaces
     */
    while (in.hasNextInt()) {
      int next = in.nextInt();

      /* Next number may not be part of this sequence
       */
      if (next > prev) {
       . . .
```

Code Citation

- Code that is not yours MUST be cited.
- Place a comment at the start of the code block that is not written by you.
- The comment should include:
  - The URL or location of where the code was retrieved from
  - The date the code was retrieved
  - The author of the code (if known)
- Place a comment at the end of the code block that is not written by you.
- Example of coding citation

```
/* The following code was taken from
 * URL: http://www,stackoverflow.com/.....
 * Retrieved on February 30, 2020
 * Author: Josephina Q. Public
 */


…


/* Cited Code Ends */
```

Avoid Complex or Duplicate Expressions

- Complex expressions or duplicate subexpressions should be broken up using local variables
- This avoids two problems:
  - Forgetting to change one subexpression when the other changes
  - Recomputing values that have already been computed (less of an issue)
- Note: Object oriented programming may lead you unwittingly to this problem
- If a statement or expression is spanning multiple lines, you should be considering breaking it down into simpler subexpressions!

Bad

```
if ((r > (b - sqrt(b * b - 4 * a * c)) / 2)) &&
    (r < (b + sqrt(b * b - 4 * a * c)) / 2)) {
  ...
}
```

Good

```
double tmp = sqrt(b * b - 4 * a * c) / 2;
double b2 = b/2;
if ((r > b2 - tmp) && (r < b2 + tmp)) {
  ...
}
```

Other Guidelines to Consider

- Place only one statement per line
- Lines should be at most 80 characters long

  Why? It’s a standard because old systems had 80 character-wide screens

General Considerations

- Follow the language’s programming idioms (conventions)
  - Naming conventions
  - Commenting
  - Variable declarations
  - Etc.

  E.g., in old-school C variables are declared at start of a function, in Java variables are declared as close as possible to where they are used. (New-school C follows C++/Java conventions)
- When working on an existing project, follow its style guidelines

  Never mix style guides!
- If deciding on which style guide to use, **use a standard one**, do not create your own, e.g.,
  - Google Java Style Guide: https://google.github.io/styleguide/javaguide.html
  - PEP8 for Python: https://www.python.org/dev/peps/pep-0008/
  - For C there are many standards, here is a common one: https://www.doc.ic.ac.uk/lab/cplus/cstyle.html

CSCI 2134 Layout and Style

- Will be looking to you to maintain a common program layout and style
- The CSCI 2134 Style Guide is the same as used in CSCI 1110
- Please review the PDF on the course web page

The Cost of Bad Code

- Hard to edit / maintain
- Missing client deadlines / windows of opportunity
- Money / time spent fixing the code
- Risk of data breach
- Loss of trust from clients / reputation
- Not scalable, extensible, adaptable
- Poor internal morale
- Low re-use
- Loss of market share

THE PROJECT I INHERITED HAS WEAK CODE. I NEED TO REWRITE IT FROM SCRATCH.
WILL THERE EVER BE AN ENGINEER WHO SAYS, "THAT LAST GUY DID A GREAT JOB. LET'S KEEP ALL OF IT"?
I'M HOPING THE IDIOT YOU HIRE TO REPLACE ME SAYS THAT.
Dilbert.com DilbertCartoonist@gmail.com
8-12-14 ©2014 Scott Adams, Inc./Dist. by Universal Uclick

Key Points

- Developer’s criteria for software quality focuses on the quality of the code itself rather than functionality
- Software quality criteria include: readability, testability, understandability, maintainability, reusability, flexibility, and portability
- As new developers, our biggest impact is in software readability and testability
- Software readability depends the logical structure of the code, the source documentation, and the layout and style

Image References

Retrieved December 29 - 31, 2019

- http://pengetouristboard.co.uk/vote-best-takeaway-se20/
- https://www.twotwentyone.net/wp-content/uploads/2013/08/USB-receptacle.jpg
- https://i.pinimg.com/originals/b5/22/38/b52238fad11b0a3ecac36fa176041d98.jpg
- https://webstockreview.net/images/clipart-hospital-money-3.png
- https://s3-production.bobvila.com/articles/wp-content/uploads/2018/04/Reasons_Electrical_Outlet_Not_Working.jpg
- https://c7.uihere.com/files/109/173/249/software-quality-assurance-quality-control-quality-management-assurance.jpg
- https://i7.pngguru.com/preview/380/91/790/hourglass-time-clock-clip-art-vector-illustration-time.jpg
- https://1001freedownloads.s3.amazonaws.com/vector/thumb/133270/neoguiri_Barrier.png
- https://thumbs.dreamstime.com/z/six-components-project-charter-components-project-charter-159700743.jpg
- https://thumbs.dreamstime.com/b/compliance-rules-regulations-guidelines-arrow-signs-words-colorful-road-directing-you-to-comply-wih-important-laws-31478130.jpg
- https://www.researchgate.net/profile/Mehmet_Celepkolu/publication/329855173/figure/fig2/AS:706489106841600@1545451537633/Pair-programming-setting-Students-look-in-different-directions-during-the-session.png
- https://www.slideshare.net/ChihyangLi/object-oriented-programming-ch3-srp-dip-isp
- https://www.clipartmax.com/png/middle/146-1467994_windows-symbol-mark-start-menu-icon-windows-8.png
- https://library.kissclipart.com/20181207/wyw/kissclipart-linux-logo-png-clipart-linux-foundation-tux-e477406d44b4921f.jpg

Retrived September 18, 2020

- https://www.explainxkcd.com/wiki/index.php/File:code_quality.png