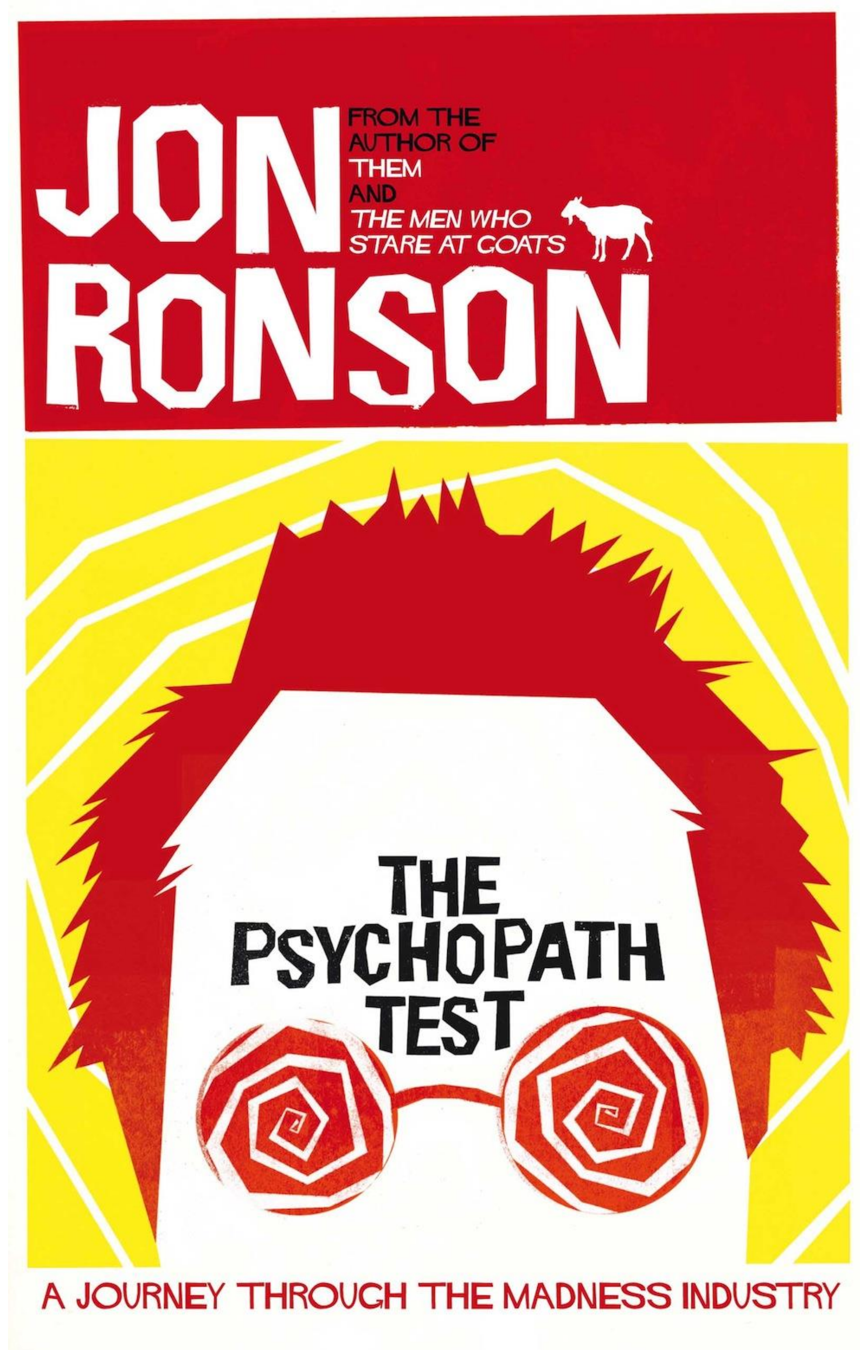
Java Programming 2 Style

Mary Ellen Foster

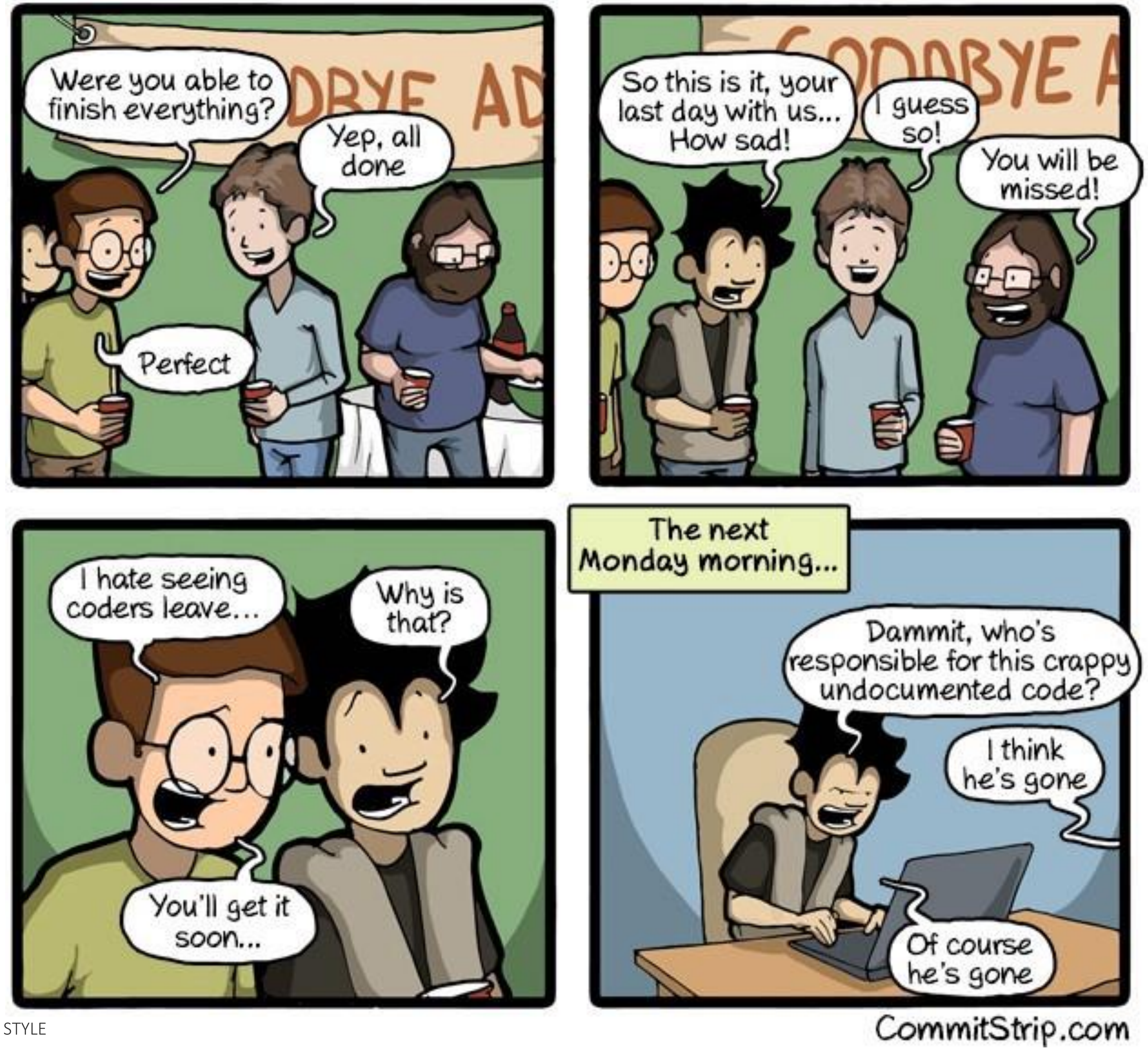
MaryEllen.Foster@glasgow.ac.uk

Photo by Kaizen Nguyễn on Unsplash


JON RONSON
FROM THE AUTHOR OF THEM AND THE MEN WHO STARE AT GOATS
THE PSYCHOPATH TEST
A JOURNEY THROUGH THE MADNESS INDUSTRY

"Always code as if the [person] who ends up maintaining your code will be a violent psychopath who knows where you live. Code for readability."

John F. Woods



GOODBYE AD
Were you able to finish everything?
Yep, all done
Perfect
So this is it, your last day with us... How sad!
I guess so!
You will be missed!
I hate seeing coders leave...
Why is that?
You'll get it soon...
The next Monday morning...
Dammit, who's responsible for this crappy undocumented code?
I think he's gone
Of course he's gone
CommitStrip.com

What is “good coding style”?

Some aspects are subjective/trivial and discussions can turn into “Bikeshedding”

- https://en.wikipedia.org/wiki/Law_of_triviality

There are generally accepted conventions though, e.g.

- Google https://google.github.io/styleguide/javaguide.html
- Apache commons https://commons.apache.org/proper/commons-net/code-standards.html
- Spring https://github.com/spring-projects/spring-framework/wiki/Code-Style

See also https://medium.com/@rhamedy/a-short-summary-of-java-coding-best-practices-31283d0167d3

Declarations

Class/interface name should start with a capital letter and be in CamelCase

Public class **MyClass** should be saved in a file called **MyClass.java**

- One public class per file

Variable/field names should be descriptive but not overly long

- **schoolId** rather than **id** or **schoolIdentificationNumber**

Methods should normally start with a verb and be in camelCase

Static fields in ALL_CAPS

Order of members in a class/interface

Group related members together

Fields first, one per line, blank lines between groups of fields them if they divide up logically

Next, all constructors

Methods after constructors

- Use a logical order – getters/setters together? Programmatically related methods?
  - *Just don’t use “the order I wrote them in”*
- Group all overloaded methods (i.e., methods with the same name) together

Indentation and spacing

Unlike Python, indentation is not **required**, but it is extremely important for readability

General rule:

- Every new block (curly brackets) should be indented one more unit
- Many style guides insist on only space characters; I personally don't care if you use tabs

Line width – something between 70-120 characters is standard

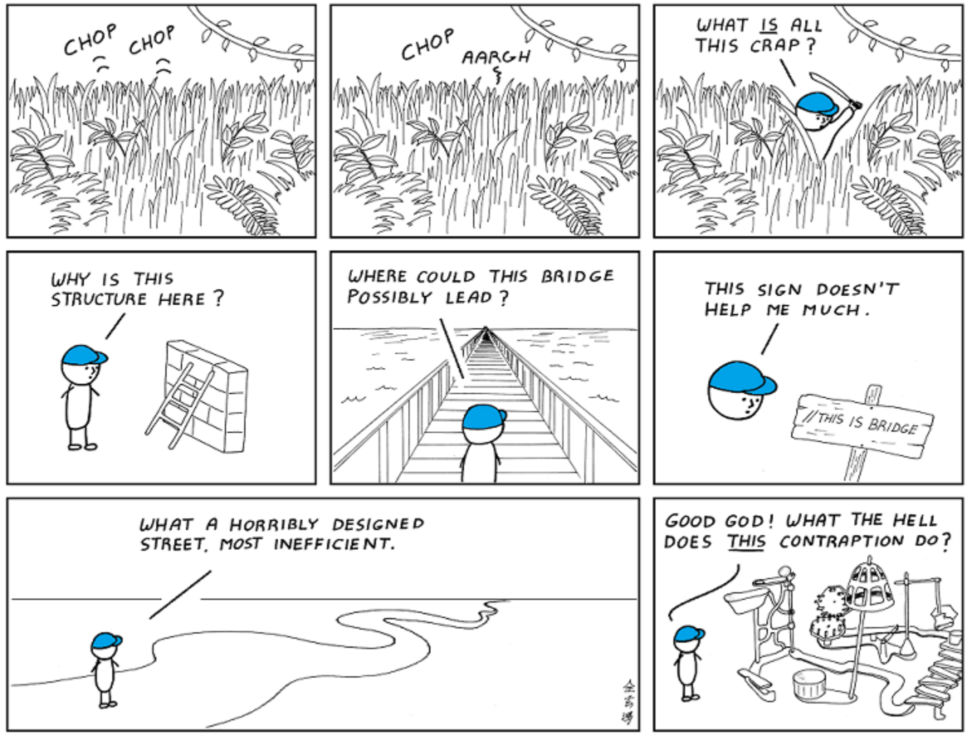
A good editor or IDE will largely manage indentation and spacing for you

Spacing between blocks of code:

- Put closely related lines together
- Leave a blank line between separate "thoughts"

Commenting

- Two types of comments
  - Implementation comments – commenting about a particular aspect of code
  - Documentation comments – describing code specification to an outside reader


CHOP CHOP
CHOP AARGH
WHAT IS ALL THIS CRAP?
WHY IS THIS STRUCTURE HERE?
WHERE COULD THIS BRIDGE POSSIBLY LEAD?
THIS SIGN DOESN'T HELP ME MUCH.
//THIS IS BRIDGE
WHAT A HORRIBLY DESIGNED STREET. MOST INEFFICIENT.
GOOD GOD! WHAT THE HELL DOES THIS CONTRAPTION DO?

I hate reading other people's code.

Implementation comments

Describing a block of code – often in conjunction with blank line before

```
  // Sanity check on extracted headers
  // If number is blank it's probably a blank line, skip it
```

Calling out or explaining some non-obvious implementation decision

```
  // Uggh, non-Collection APIs
  // Skip it, Mac weirdness
```

Shouldn't be trivial

```
  i++; // Add one to i
```

You don't need a comment on every line!

Where do you put implementation comments?

Usually, implementation comments go above the block of code that they refer to

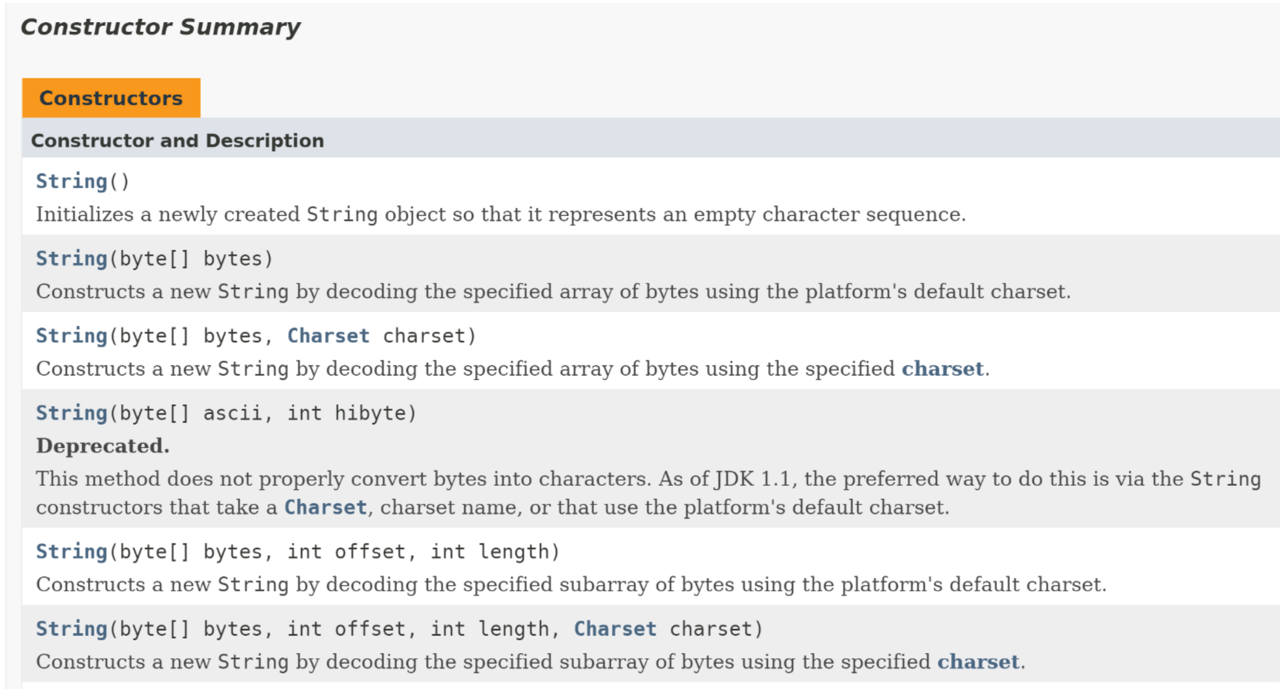
```
// Sanity check on extracted headers
if (parts.size() == 0) {
    JOptionPane.showMessageDialog(null, "No headers found in “
        + jfc.getSelectedFile(), "No headers found", JOptionPane.WARNING_MESSAGE);
    wb.close();
    System.exit(0);
}
```

Some people like to put them on the same line as the code they refer to

(I kind of hate this but won’t penalise you for it ☺)

Documentation comments: Javadoc

- Processes Java source files and generates HTML documentation
  - This is what generates the Java class documentation at https://docs.oracle.com/javase/8/docs/api/
- Reflects the structure of the source files
- Also includes any content in specially-formatted comments



Constructor Summary

Constructors

Constructor and Description
`String()` Initializes a newly created `String` object so that it represents an empty character sequence.
`String(byte[] bytes)` Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.
`String(byte[] bytes, Charset charset)` Constructs a new `String` by decoding the specified array of bytes using the specified **charset**.
`String(byte[] ascii, int hibyte)` **Deprecated.** This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a `Charset`, charset name, or that use the platform's default charset.
`String(byte[] bytes, int offset, int length)` Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.
`String(byte[] bytes, int offset, int length, Charset charset)` Constructs a new `String` by decoding the specified subarray of bytes using the specified **charset**.

Javadoc comments

- Surrounded by `/** … */`
  - Blue rather than green in Eclipse
- Refer to the class member that comes directly below them
  - Classes
  - Interfaces
  - Constructors
  - Fields
  - Methods
- Can contain HTML

```
/** Class Description of MyClass */
public class MyClass {
        /** Field Description of myIntField */
        public int myIntField;
        /** Constructor Description of MyClass() */
        public MyClass() {
                // Do something ...
        }
}
```

https://students.cs.byu.edu/~cs240ta/fall2012/tutorials/javadoctutorial.html

Javadoc tags

Tags: keywords recognised by Javadoc to identify information

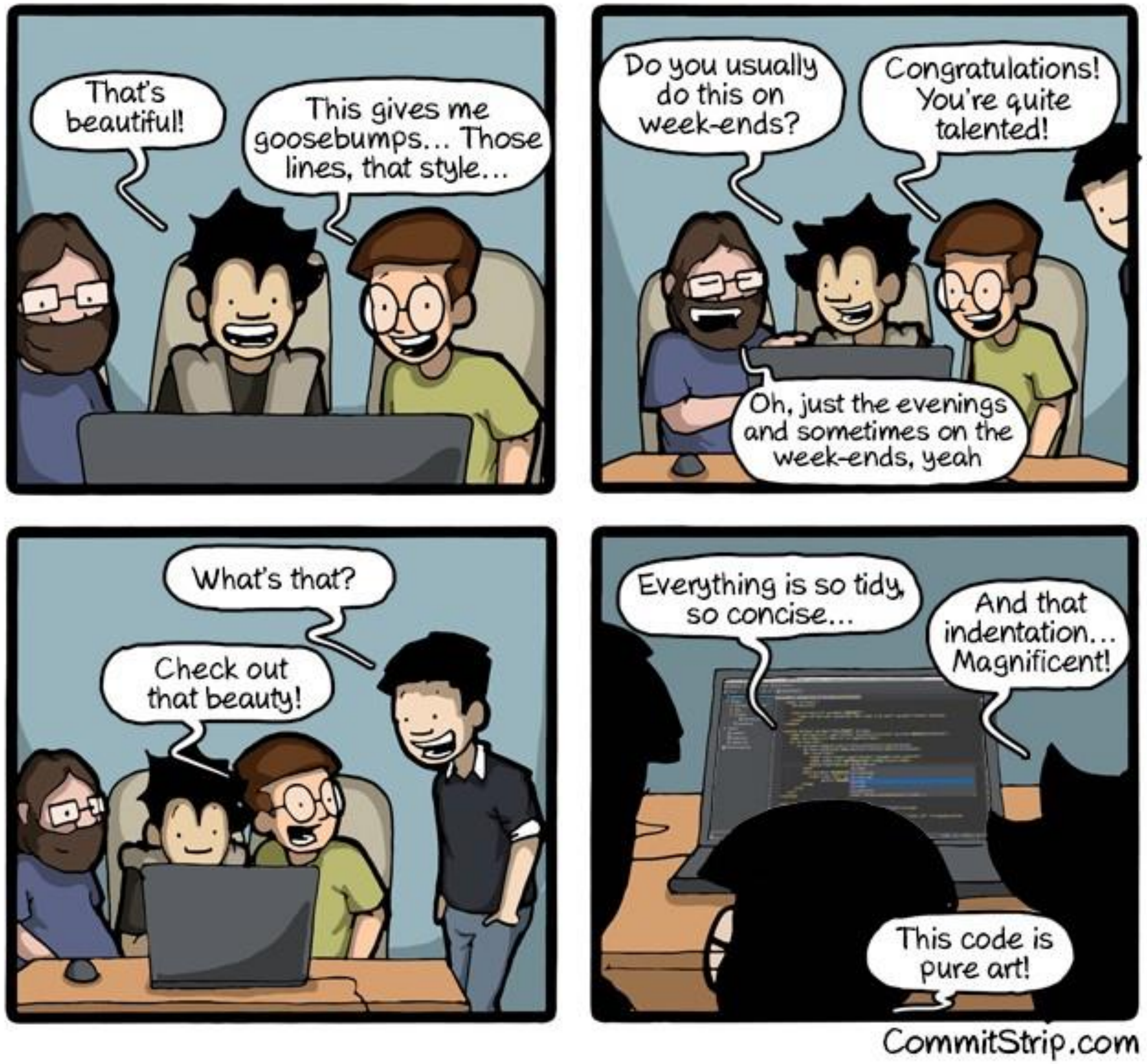
- @author *name*
- @version *version*
- @param *name description*
- @return *description*
- @throws *exception description*
- @see *other-class*

```
/**
* <h1>Hello, World!</h1>
* The HelloWorld program implements an application that
* simply displays "Hello World!" to the standard output.
* <p>
* Giving proper comments in your program makes it more
* user friendly and it is assumed as a high quality
code.
*
*
* @author  Zara Ali
* @version 1.0
*/

public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World!");

    }
}
```

http://www.tutorialspoint.com/java/java_documentation.htm

Another example (CreditCard.java)

```java
/**
 * Attempts to make a charge to the credit card.
 *
 * @param amount
 *            The amount to charge
 * @return true if the card still has enough room to make the charge, and
 *         false if not
 * @throws Exception
 *             if the amount to charge is invalid (i.e., not positive)
 */
public boolean charge(double amount) throws Exception {
    if (amount <= 0) {
        throw new Exception("Invalid charge amount: " + amount);
    }

    if ((amount + currentBalance) <= creditLimit) {
        currentBalance += amount;
        return true;
    } else {
        return false;
    }
}
```


That's beautiful!
This gives me goosebumps... Those lines, that style...
Do you usually do this on week-ends?
Congratulations! You're quite talented!
Oh, just the evenings and sometimes on the week-ends, yeah
What's that?
Check out that beauty!
Everything is so tidy, so concise...
And that indentation... Magnificent!
This code is pure art!
CommitStrip.com