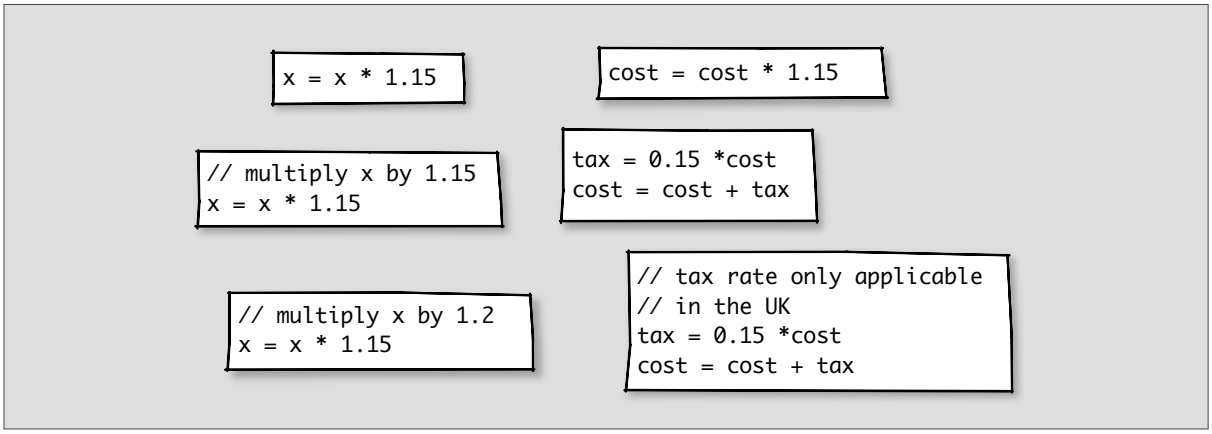
Writing Readable Code

ippo 2021-2022



Here are two code samples. These two implementations are intended to have the same functionality. One of them "works", but the other has a bug. If you had to work with this code, which one would you choose?

```
class Main{static $1 $_;class $1{void _$(String $_){System.//
out.print($_);}$1(){x();}void x(){int x,$,_$,$$,__,a=(1<<5),
b=100,c=12,bc=a*c;b-=1<<4;while(bc>0){for($$=_$=x=__=$=(int)
b;_$>($$-(1<<2));_$(""+(char)(_$)),_$-=1<<1)for(x=$=__-9;x>(
$-6);x-=1<<1,_$(""+(char)(_$!=b?x:a)));char S$=(char)(b+c+1)
;_$("te"+S$+(char)(c+(int)S$));bc--;}}}Main(){$_=new $1();}
public static void main(String []$){Main b=new Main();}}
```

```
public class Main {
    public static void main(String[] args) {
        for (int i=1; i<=384; --i) {
            System.out.print("   TIGERTeam");
        }
    }
}
```

In the real world, "working" code is not the only critera. If it is not easy for other people to read and understand, easy to extend later, and sufficiently clear that there are no hiding places for obscure bugs, then it is not useful. This is the reason why clear, readable code is a core requirement for the IPPO course.

Different organisations and projects have different conventions and "standards" to ensure that their code is consistent. In practice, you will need to adapt your style to fit in with the project that you are working on. We don't enforce any particular style for this course, but you must make sure that your style is consistent and clear. The following tips might help you think about the readability of your code:

1 Comments

In general, the code itself should be sufficiently clear that it does not require additional comments on every statement. Too many comments can make the code more difficult to read. Comments which are

"out of sync" with the code are particularly bad! It is often better to just write clear code. Compare the range of comments and code at the start of the document. Which of these are helpful?

You will need to judge for yourself what you think makes the code as clear as possible to a (similarly-experienced) reader. Some typical places where comments might be necessary are:

- At the top of each file to explain the overall purpose of the file/class.
- To explain the reason behind a statement (rather than what the statement does). For example: "Why does this statement add 10% to the tax" ?
- At the start of a method where the purpose and/or arguments may not be obvious.
- On variable declarations to explain the purpose of the variable.

Make sure that you comments are meaningful and up-to-date: if you copy and edit a template file, remember to delete/edit any comments which are no longer appropriate!

Javadoc: Javadoc is a system for automatically generating documentation from structured comments in the source code. The online documentation for the standard Java libraries, and the IPPO assignment libraries are automatically generated in this way. This is particularly useful for documenting libraries of public methods which are intended for use by other people.

For the IPPO assignments, it is sufficient to provide consistent and clear comments (see above), and Javadoc is not required. However, if you would like to know more about Javadoc, or use it in your own code, the The BlueJ book has an appendix on Javadoc. Gradle has a Javadoc task[1] to generate HTML from Javadoc , and the Oracle website[2] has further information on Javadoc.

2 Variable names

Meaningful variable names go a long way towards helping others understand your code without additional comments. Choose these carefully, and be consistent in your use of case - we strongly recommend that you use the following conventions:

Methods: lowerCamelCase (`addSubject`, `getPicture`)

Classes: UpperCamelCase (`MyCacheProxy`, `Controller`)

Packages: lowercase (`model`)

Constants: CONSTANT_CASE (`SPEED_OF_LIGHT`, `PI`)

This does not mean that you shouldn't use short names where appropriate: `x` and `y` are perfectly good names for the coordinates of a point, if there is only one point and the variables are used within a small scope. Similarly, `i` is a perfectly good name for a counter if the loop is short and the purpose is obvious. If the scope is larger, and the reader has to search for the variable declaration, then you probably need a more meaningful name.

3 Layout

The layout of the code (particularly the indentation) is critical for the reader to be able to understand the logic of your program. If conditionals or loops are incorrectly indented, then it is very easy for the reader to misinterpret the code - and this is very bad.

[1] https://docs.gradle.org/current/dsl/org.gradle.api.tasks.javadoc.Javadoc.html
[2] http://download.oracle.com/javase/6/docs/technotes/tools/solaris/javadoc.html#documentationcomments

This bug[3] compromised the SSL encryption in all iPhones in 2016:

```
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
  goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
  goto fail;
  goto fail;                                                    ← Oops!
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
```

IntelliJ attempts to keep the code automatically formatted as you are editing. But it is possible to create badly formatted code, and Code » Reformat Code can be used to explicitly reformat sections of code.

4 Structure

All of the above suggestions refer to the small-scale syntax and layout of the code. When reading a larger program, the most difficult challenge is usually to understand how the code is structured and where to find the particular part of the code that performs a certain function. Writing code which is clear in this way takes more experience, and involves a consideration of the structure of the code itself, rather than just the surface features. Here are some things that you should consider ...

- Make sure that the classes are as cohesive and loosely coupled as possible.
- Make sure that each class is clearly commented with its purpose.
- Avoid long and complicated methods or control structures - factor out their contents into individual methods with meaningful names and parameters.

[3] http://blog.languager.org/2016/06/break-is-goto-in-disguise.html