



WRITING CLEAN CODE

By Pablo Restrepo

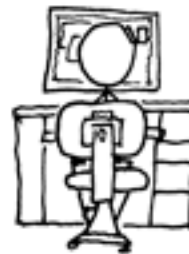
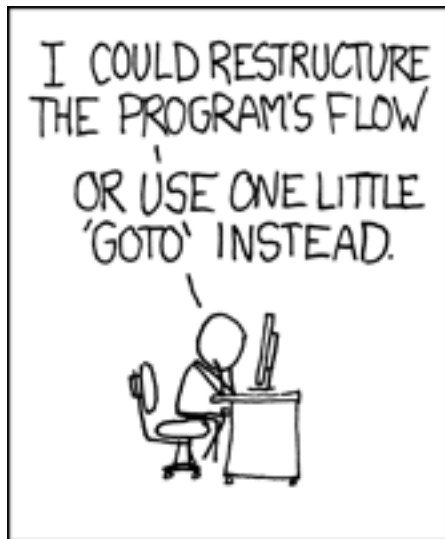
Based on the book
"Clean Code" by Uncle
Bob





2. FUNCTIONS

Functions





Functions

Let's Try to understand what this code does
in less than 3 minutes

<https://github.com/prestrepoh/Clean-Code-Course/blob/master/2-functions/longfunction.java>





Functions

It's very difficult to understand the method when it has odd function calls and doubly nested if statements controlled by flags.

What about this refactored version of the code?

<https://github.com/prestrepoh/Clean-Code-Course/blob/master/2-functions/longfunctionrefactored.java>





Functions

Why is function 2 easier to understand?

How can we make a function communicate
it's intent?





Small!


The first rule of functions is that they should be small. The second rule of functions is that they should be **smaller than that**

Functions should hardly ever be 20 lines long

According to that. Our example function should be shortened to:

<https://github.com/prestrepoh/Clean-Code-Course/blob/master/2-functions/longfunctionrefactoredshortened.java>





Blocks and Indenting

Blocks between if, else or while, etc statements should be one line long (they probalby should be a function call).

A function should not be large enough to hold nested structures.





Do One thing

**FUNCTIONS SHOULD DO ONE THING. THEY
SHOULD DO IT WELL.
THEY SHOULD DO IT ONLY.**



only do
one
thing

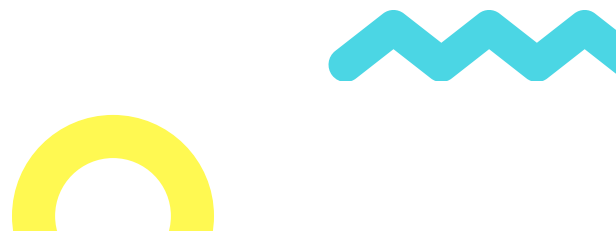


Do One thing

You may think that the refactored version of our example does three things:

1. Determining whether the page is a test page.
2. If so, including setups and teardowns.
3. Rendering the page in HTML.

But we should note that the three steps of the function are one level of abstraction below the stated name of the function.

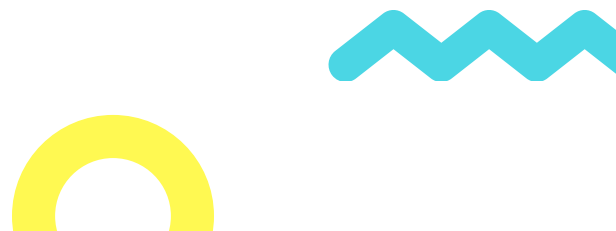




Do One thing

If a function does only those steps that are one level below the stated name of the function, then the function is doing one thing.

In our shortened example, we could extract the if statement into a function named `includeSetupsAndTearardownsIfTestPage`, but that simply restates the code without changing the level of abstraction.

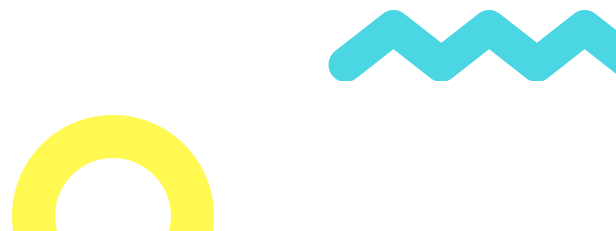




Do One thing

You can know if a function does more than one thing if you can extract another function from it with a name that is not merely a restatement of its implementation.

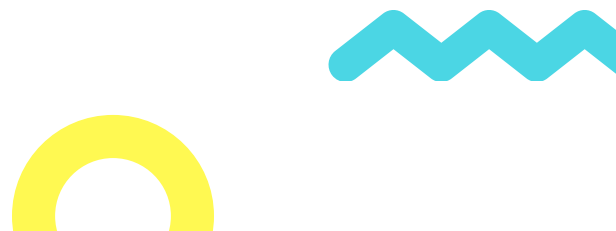
When a function is divided into sections such as declarations, initializations, and sieve, it is an obvious symptom of doing more than one thing. Functions that do one thing cannot be reasonably divided into sections.





One Level of Abstraction per Function

In order to make sure our functions are doing
“one thing,” we need to make sure that the
statements within our function are all at the
same level of abstraction





One Level of Abstraction per Function

In the first version of our example, there are concepts in there that are at a very high level of

abstraction, such as ***getHtml()***; others that are at an intermediate level of abstraction, such

as: ***String pagePathName =***

PathParser.render(pagePath); and still others that are remarkably

low level, such as: ***.append("\n")***.

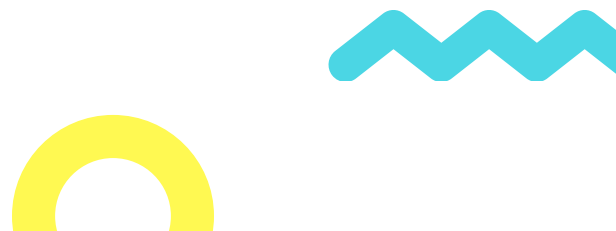




The Stepdown Rule

We want our method to read as a story.

We want to be able to read the program as
though it were a set
of TO paragraphs

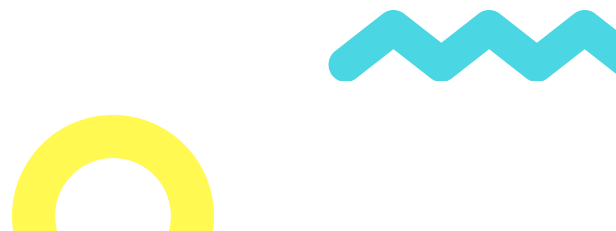




Switch Statements

By nature, switch statements do more than one thing. Unfortunately we can't always avoid switch statements.

we can make sure that each switch statement is buried in a low-level class and is never repeated with ***polymorphism***.





Switch Statements

Consider the following code:

<https://github.com/prestrepoh/Clean-Code-Course/blob/master/2-functions/1-switch-statements/badswitchcase.java>

What's wrong with it?.





Switch Statements

It is large and will grow when more employee types are added.

It does more than one thing.

It violates the Single Responsibility Principle.

It violates the Open Closed Principle.

there are an unlimited number of other functions that will have the same structure. For example:

isPayday(Employee e, Date date),

or

deliverPay(Employee e, Money pay)

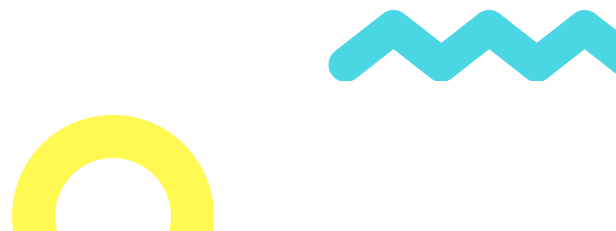




Switch Statements

The solution: Bury the switch statement in the basement of an ABSTRACT FACTORY, and never let anyone see it.



<https://github.com/prestrepoh/Clean-Code-Course/blob/master/2-functions/1-switch-statements/employeeabstractfactory.java>





Switch Statements

General rule for switch statements is that they
can be tolerated if they appear
only once, are used to create polymorphic
objects, and are hidden behind an inheritance



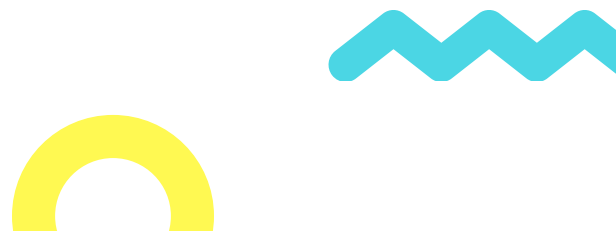




Use Descriptive Names

Choose a descriptive name that says what the function does.

Don't be afraid to make a name long. A long descriptive name is better than a short enigmatic name.

A long descriptive name is better than a long descriptive comment.





Function Arguments

Arguments are hard to follow.

The ideal number of arguments for a function is zero, followed by one, followed by two.

Three arguments should be avoided when possible, and more that should never be used.





Common Monadic Form

Pass arguments when:

Asking questions about it: *boolean
fileExists("MyFile")*.

Transforming it into something else and
returning it: ***InputStream fileOpen("MyFile")***
transforms a file name String into an
InputStream return value.

Capturing an event: *void
passwordAttemptFailedNtimes(int attempts)*.





Common Monadic Form

Try to avoid any monadic functions that don't follow these forms, for example, ***void includeSetupPageInto(StringBuffer pageText).***

NEVER USE OUTPUT ARGUMENTS!.





Flag Arguments

Flag arguments are ugly. It announces that the function does more than one thing (if flag true do something if not do something else).





Dyadic functions

Common used dyadic functions like ***assertEquals(expected, actual)*** are problematic.

It makes sense to use dyadic functions in the case:

Point p = new Point(0,0);






Dyadic functions

You should use mechanisms to avoid passing more than one parameter. For Example:

Make the
writeField method a member of outputStream so
that you can say ***outputStream.
writeField(name)***

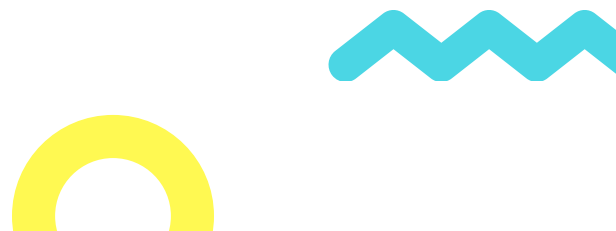






Triads

Functions that take three arguments are significantly harder to understand than dyads

Make the
writeField method a member of outputStream so
that you can say ***outputStream.
writeField(name)***





Argument Objects

*Circle makeCircle(double x, double y, double
radius);*

VS

Circle makeCircle(Point center, double radius);

There are two arguments of the function that
belong to the same concept.

