**[Double Brace Initialization](http://c2.com/cgi/fullSearch?search=DoubleBraceInitialization)**

One of the [JavaIdioms](http://wiki.c2.com/?JavaIdioms). [JavaLanguage](http://wiki.c2.com/?JavaLanguage) doesn't have a convenient literal syntax for collections (lists, maps, sets, etc.). This makes creating constant collections or passing collections to functions quite laborious. Every time you have to

1. Declare a variable for a temporary collection
2. Create a new empty collection and store a reference to it in the variable
3. Put things into the collection
4. Pass the collection to the method

E.g. To pass a set to a method:

Set<String> validCodes = new HashSet<String>();

validCodes.add("XZ13s");

validCodes.add("AB21/X");

validCodes.add("YYLEX");

validCodes.add("AR2D");

removeProductsWithCodeIn(validCodes);

Or to initialize a set of constants:

private static final Set<String> VALID\_CODES = new HashSet<String>();

static {

validCodes.add("XZ13s");

validCodes.add("AB21/X");

validCodes.add("YYLEX");

validCodes.add("AR2D");

}

But... you can create and initialize a new collection *as an expression* by using the "double-brace" syntax:

E.g.

private static final Set<String> VALID\_CODES = new HashSet<String>() {{

add("XZ13s");

add("AB21/X");

add("YYLEX");

add("AR2D");

}};

Or:

removeProductsWithCodeIn(new HashSet<String>() {{

add("XZ13s");

add("AB21/X");

add("YYLEX");

add("AR5E");

}});

The first brace creates a new [AnonymousInnerClass](http://wiki.c2.com/?AnonymousInnerClass), the second declares an instance initializer block that is run when the anonymous inner class is instantiated. This type of initializer block is formally called an "instance initializer", because it is declared within the instance scope of the class -- "static initializers" are a related concept where the keyword static is placed before the brace that starts the block, and which is executed at the class level as soon as the classloader completes loading the class (specified at <http://docs.oracle.com/javase/specs/jls/se5.0/html/classes.html#8.6>) The initializer block can use any methods, fields and final variables available in the containing scope, but one has to be wary of the fact that initializers are run before constructors (but not before superclass constructors).

This only works for non-final classes because it creates an anonymous subclass. Obviously, this is not limited to collections; it can be used to initialize any kind of object -- for example Gui objects:

add(new JPanel() {{

setLayout(...);

setBorder(...);

add(new JLabel(...));

add(new JSpinner(...));

}});

The instance of the anonymous class that you have created contain a synthetic reference to the enclosing object. If you serialise the collection you will also serialise everything in the outer class.

This idea is incompatible with one popular way to implement the *equals(Object o)* method. Assume class *Example* has this method:

public boolean equals(final Object o) {

if (o == null) {

return false;

} else if (!getClass().equals(o.getClass())) {

return false;

} else {

Example other = (Example) o;

// Compare *this* to *other*.

}

}

Then, objects created by [DoubleBraceInitialization](http://wiki.c2.com/?DoubleBraceInitialization) will never be equal to objects created without it. I would never use this for any class that needs a nontrivial *equals(Object)*method. Collection classes should be fine. So should the JPanel instance above. -- [EricJablow](http://wiki.c2.com/?EricJablow)

*It would be nice if Java only created and initialised an instance, not create a new class, for double brace initialisation and any anonymous "class" that does not add fields or override methods.*

An idiomatic alternative, that doesn't require such heavy handed use of anonymous inner classes, is to use the constructor arguments to collections that accept another collection as the source data.

List<String> myList = new [ArrayList](http://wiki.c2.com/?ArrayList)<String>(Arrays.asList("One", "Two", "Three"));

Although, this only works with collections that can be represented as a single array -- it doesn't work for maps.

Or, use a [VarargsCollectionFactoryMethod](http://wiki.c2.com/?VarargsCollectionFactoryMethod).

>>Although this only works for collections<<< True but you can use Commons ArrayUtils.toMap(new MapUtils.toMap(new String[][] {{

{"RED", "#FF0000"},

{"GREEN", "#00FF00"},

{"BLUE", "#0000FF"}});

I call [GreencoddsTenthRuleOfProgramming](http://wiki.c2.com/?GreencoddsTenthRuleOfProgramming) for having/trying to use a programming language for config info data-entry. A little bit in code is understandable. A lot is wrong-tool-smell. -t

Last edit June 15, 2014, See [github](https://github.com/WardCunningham/remodeling" \t "_blank) about remodling.