>

CodeKata

Because experience is the *only* teacher

- RSS

Search

Navigate…

- PragDave
- Kata
- Archives

Kata04: Data Munging

Martin Fowler gave me a hard time for Kata02, complaining that it was yet another single-function, academic exercise. Which, or course, it was. So this week let's mix things up a bit.

Here's an exercise in three parts to do with real world data. Try hard not to read ahead—do each part in turn.

Part One: Weather Data

In weather.dat you'll find daily weather data for Morristown, NJ for June 2002. Download this text file, then write a program to output the day number (column one) with the smallest temperature spread (the maximum temperature is the second column, the minimum the third column).

Part Two: Soccer League Table

The file football.dat contains the results from the English Premier League for 2001/2. The columns labeled 'F' and 'A' contain the total number of goals scored for and against each team in that season (so Arsenal scored 79 goals against opponents, and had 36 goals scored against them). Write a program to print the name of the team with the smallest difference in 'for' and 'against' goals.

Part Three: DRY Fusion

Take the two programs written previously and factor out as much common code as possible, leaving you with two smaller programs and some kind of shared functionality.

Kata Questions

- To what extent did the design decisions you made when writing the original programs make it easier or harder to factor out common code?
- Was the way you wrote the second program influenced by writing the first?
- Is factoring out as much common code as possible always a good thing? Did the readability of the programs suffer because of this requirement? How about the maintainability?

Posted by Dave Thomas (@PragDave) Dec 26th, 2013

Tweet 3 g+1 0

« Kata05: Bloom Filters Kata03: How Big? How Fast? »

Comments

5 Comments pragdave 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

axonneuron · 6 months ago

Seems like the base functionality in each is to find the least difference (?) between pairs of values. Base class with the base functionality, derived classes with some overridden functions for a solution specific to each problem? Anyway, thanks for the excercise.

Reply · Share ›

Michael Küsters · 7 months ago

One might reasonably argue whether refactoring 109 characters of total execution script into method libraries is "refuctoring", so mhm....

Refactoring 30 bytes of sourcecode into caller+module doesn't look like a meaningful advantage - less so since 2 sources are less readable than 1. Treasure hunt pattern for the win!

Here is an open question:
At how many bytes do you start refactoring?
Would you refactor 3 duplicate bytes of sourcecode just because they are the same?

Reply · Share ›

Tyler Seaton · 10 months ago

I first focused on answering both questions separately. From there I took some quick and dirty notes on what was common between both approaches (although just by looking at it I could see where I had repeated myself). Next I looked to condense/remove/refactor.

I'm with Fred below: if you are cutting and pasting code you are most likely "doing it wrong". That being said I find it is almost always better to get something working before diving into refactoring. Premature refactoring is about as bad as repeating yourself.

• Reply • Share ›

Fred_Scuttle • a year ago

In answer to your questions:
- The design decisions I made for the first program, ignoring header lines, a method for removing extraneous characters from a numeric fields, made the second much easier.
- I simply modified the first program to address the second problem. Most of the same code was applicable.
- I'm a big believer in continuous refactoring. Number of lines to skip as well as field numbers were extracted as Java static final constants. I left other methods in place, such as the string trimming. That way, if the file format changes then the code modifications should be limited to changing the constants. If you're going to be using (essentially) the same code for different purposes then I believe that you should break out the common code into separate methods or even classes. I often create utility classes for specific functions, whether it's document conversion or cryptographic routines. I can readily move these classes to different projects or easily and conveniently reuse the methods as the application grows.
But it's sometimes hard to break junior programmers of the "cut and paste" habit. I always tell them that if they're cutting and pasting then they're doing something wrong. Invest the time to break out the common code into a separate method and ALWAYS document appropriately. We've actually got a target in our ant build.xml called javadoc. The idea is that, if everyone plays by the rules and documents code properly, then we have the potential for code re-use. And if it means that sometimes you have to use over-loaded methods then there's absolutely nothing wrong with that in my book.

• Reply • Share ›

Michael Marcal ➦ Fred_Scuttle • 3 months ago

Agree with Michael Kusters above. I did both in Python. Without trying to be parsimonious in coding, the football script was 20 lines; the weather script 13. Not sure factoring code makes a lot of sense here.
The crux of the problem for me was: how to use the initial format line to drive the locations of the fields in parsing the data in the subsequent ones. I didn't do that in the weather example, so that code had hard-coded numbers in the algo. The code for the football example (Sigh, my beloved Tottenham always the middle of the pack) had no "numbers" in the code.

• Reply • Share ›

ALSO ON **PRAGDAVE** WHAT'S THIS?

Elixir: State Machines, Metaprogramming, and Generating Tests

1 comment • 8 months ago

Avat **Bill Christian** — As a newcomer to elixirlang, Enum.reduce and Stream.unfold continue to confuse me. I am working through some

Agile Is Dead (Long Live Agility)

188 comments • a year ago

Avat **Pattern-chaser** — Mr Kutz, I fear the industry is as bad or worse than Dave said. Many

Kata05: Bloom Filters

4 comments • a year ago

Avat **Manh** — But In the paper ,It different from conventional bloom filters You've read the paper "Multi-dimensional Range Query for

Kata03: How Big? How Fast?

5 comments • a year ago

Avat **lazel** — I can't find the solution, so I'll write mine here. I'm *really* curios to know the

Recent Posts

- CodeKata
- CodeKata: How It Started
- Kata, Kumite, Koan, and Dreyfus
- Kata01: Supermarket Pricing
- Kata02: Karate Chop
- Kata03: How Big? How Fast?
- Kata04: Data Munging
- Kata05: Bloom Filters
- Kata06: Anagrams
- Kata07: How'd I Do?
- Kata08: Conflicting Objectives
- Kata09: Back to the Checkout
- Kata10: Hashes vs. Classes
- Kata11: Sorting It Out
- Kata12: Best Sellers
- Kata13: Counting Code Lines
- Kata14: Tom Swift Under the Milkwood
- Kata15: A Diversion
- Kata16: Business Rules
- Kata17: More Business Rules
- Kata18: Transitive Dependencies
- Kata19: Word Chains
- Kata20: Klondike
- Kata21: Simple Lists