THE CODING DOJO HANDBOOK

*a practical guide to creating a space where **good** programmers can become **great** programmers*

Emily Bache

Foreword by Robert C. Martin

The Coding Dojo Handbook

a practical guide to creating a space where good programmers can become great programmers

Emily Bache

ISBN 978-91-981180-0-1

Section 4: Kata Catalogue

There are many, many code katas, and this catalogue is in no way exhaustive. These are some of my favourites, and ones which I've found to work well in the context of a coding dojo.

What is a Code Kata?

A Code Kata is a small, fun problem that shouldn't take you more than an hour or two to solve in your favourite programming language. The rule is that you must repeat the exercise, and every time try to improve the way you solve the problem. Not just the code you end up with, but the process by which you get to it.

I don't think learning a code kata has anything to do with learning a sequence of keystrokes or perfectly imitating some kind of "master" programmer. That's where the analogy with Karate breaks down! When you "know" a kata, that means that solving the actual problem no longer presents any difficulty to you, and you can concentrate on improving all the other aspects of *how* you solve it. You'll be able to try out a variety of approaches: object oriented, functional languages, big tests, small tests, another order of tests, with and without faking it, refactoring at this point or that point, different datastructures, algorithms, names… Every time you do the kata, you can try out something new, or make a small improvement to an approach you've used before.

Dojo Disaster: The Architect's Kata

Emmanuel Gaillot recounted for me an incident when somebody new turned up to the Paris dojo. He described himself as a "software architect", and he suggested that not all katas need involve coding. He instead proposed a "design" kata. The group discussed the idea, and the fact that they'd set up the dojo as a place where you learn by **coding** infront of others. On the other hand, someone suggested that in order to really understand a rule, maybe you should break it and see what you can learn.

So they decided to take up the architect's suggestion, and spent an evening drawing boxes and arrows. It didn't turn out so well. As Emmanuel put it: "It was excruciating!". Everyone agreed it was not fun at all. So they kept the rule about coding - in fact, all the Katas in this catalogue involve writing code.

So I agree with Emmanuel, (see the Sidebar "The Architect's Kata"), a code kata must also involve writing actual code. And tests!

How to choose a good Kata for your dojo

The most important thing is to choose a Kata you will enjoy doing! Flip through the catalogue and pick out any topics that look interesting. Have a look at the section “Contexts to use this Kata” for an idea of what you might learn from it. If there is a skill you’re working on, there is some advice in the previous section “Teaching & Learning In the Dojo”, with suggestions of which Katas are particularly useful.

About this Catalogue

Each Kata has an explanation of the problem to be solved, and links to where you can download starting code (if applicable). In addition, I’ve added some suggestions to help you get the most out of the kata, and to choose one appropriate for your context.

Additional discussion points for the Retrospective

After you’ve done the kata, these questions might prompt interesting discussion. (You might be having a great discussion anyway, of course!) When I’m facilitating a dojo, I often find the retrospective is the hardest part. I can see that the group has learnt lots through doing the Kata, but I don’t always know how to get people talking about it. That’s why I’ve written these extra notes, to remind me of some questions that might spark good discussion.

Ideas for after the Dojo

If you’ve done this kata in a dojo, you might be inspired to try it again by yourself at home. Here are some ideas for how to extend the kata or vary it in some way, so you get the most out of it. If several dojo participants continue to work on a kata after the dojo, you can go online to share code snippets, ideas and links, and to continue to discuss what was said in the meeting. Alternatively you could share what you’ve learnt at the next dojo meeting.

Contexts to use this Kata

If you’re in a particular situation, any individual kata might be more or less suitable. This section should help you to choose a good Kata, and help you prepare for your dojo meeting.

Kata: FizzBuzz

Imagine the scene. You are eleven years old, and in the five minutes before the end of the lesson, your Maths teacher decides he should make his class more "fun" by introducing a "game". He explains that he is going to point at each pupil in turn and ask them to say the next number in sequence, starting from one. The "fun" part is that if the number is divisible by three, you instead say "Fizz" and if it is divisible by five you say "Buzz". So now your maths teacher is pointing at all of your classmates in turn, and they happily shout "one!", "two!", "Fizz!", "four!", "Buzz!"… until he very deliberately points at you, fixing you with a steely gaze… time stands still, your mouth dries up, your palms become sweatier and sweatier until you finally manage to croak "Fizz!". Doom is avoided, and the pointing finger moves on.

So of course in order to avoid embarrassment in front of your whole class, you have to get the full list printed out so you know what to say. Your class has about 33 pupils and he might go round three times before the bell rings for breaktime. Next maths lesson is on Thursday. Get coding!

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Sample output:

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
```

… etc up to 100

Additional discussion points for the Retrospective

- Is the code you have written clean? Are there any smells?
- Did you refactor throughout or do it all at the end?
- What if a new requirement came along that multiples of seven were “Whizz”? Could you add that without editing the existing code? (Cue discussion of the Open-Closed Principle)

Ideas for after the Dojo

- When you’ve got it all working for “Fizz” and “Buzz”, add “Whizz” for multiples of seven
- Then add “Fizz” also for all numbers containing a 3 (eg 23, 53)

Contexts to use this Kata

I find this an excellent kata for introducing beginners to TDD. It’s pretty straightforward to choose the order of test cases, work in small steps, and complete the whole exercise still leaving time for a decent retrospective.

Kata: Tennis

Tennis has a rather quirky scoring system, and to newcomers it can be a little difficult to keep track of. The Tennis Society has contracted you to build a scoreboard to display the current score during tennis games. The umpire will have a handset with two buttons labelled "player 1 scores" and "player 2 scores", which he or she will press when the respective players score a point. When this happens, a big scoreboard display should update to show the current score. (When you first switch on the scoreboard, both players are assumed to have no points). When one of the players has won, the scoreboard should display which one.

Your task is to write a "TennisGame" class containing the logic which outputs the correct score as a string for display on the scoreboard. You can assume that the umpire pressing the button "player 1 scores" will result in a method "wonPoint("player1")" being called on your class, and similarly wonPoint("player2") for the other button. Afterwards, you will get a call "getScore()" from the scoreboard asking what it should display. This method should return a string with the current score. *(Note: do modify the method names to match the idiom for your programming language)*

You can read more about Tennis scores here[1] which is summarized below:

1. A game is won by the first player to have won at least four points in total and at least two points more than the opponent. The score is then "Win for player1" or "Win for player2"
2. The running score of each game is described in a manner peculiar to tennis: scores from zero to three points are described as "Love", "Fifteen", "Thirty", and "Forty" respectively.
3. If at least three points have been scored by each player, and the scores are equal, the score is "Deuce".
4. If at least three points have been scored by each side and a player has one more point than his opponent, the score of the game is "Advantage player1" or "Advantage player2".

The Tennis Society has agreed that Sets and Matches are out of scope, so you only need to report the score for the current game. However, they have requested another feature with lower priority. They want to be able to change the name of the players from "player1" to "Björn Borg" and "player2" to "John McEnroe". This feature has been categorized "Nice to have", or, more accurately, "in your dreams"!

Tennis Refactoring Kata

Imagine you work for a consultancy company, and one of your colleagues has been doing some work for the Tennis Society. The contract is for 10 hours billable work, and your colleague has spent 8.5 hours working on it. Unfortunately he has now fallen ill, although he says he has

[1]http://en.wikipedia.org/wiki/Tennis#Scoring

completed the work, and the tests all pass. Your boss has asked you to take over and spend an hour or so on it so she can bill the client for the full 10 hours. She instructs you to tidy up the code a little and perhaps make some notes so you can give your colleague some feedback on his chosen design.

There are three scenarios for this refactoring kata - imagine three different consultancy companies each with their own solution to the problem. I suggest you start with the first version of the code. When you've got that looking beautiful, start over with the second and third versions.

What is nice about this Kata is that the tests are almost exhaustive, and fast to run, so any mistakes you make while refactoring should be very obvious. You should not need to change the tests, only run them often as you refactor. The code is available on github[2], for several popular programming languages.

I also recommend that if you're doing this as a refactoring kata, that you use a tool to record your session [3], so you can review how large steps you took. The aim is for as small as possible, with as few refactoring mistakes as possible.

Additional discussion points for the Retrospective

- Is the code you have ended up with clean? Are there any smells?
- Are your tests exhaustive?
- Does your code express the scoring rules of Tennis in a readable manner?

Refactoring version

- How did it feel to work with such fast, comprehensive tests?
- Did you make mistakes while refactoring that were caught by the tests?
- If you used a tool to record your test runs, review it. Could you have taken smaller steps?
- Did you ever make a refactoring mistake and then back out your changes? How did it feel to throw away code?
- If you never backed out any refactoring mistakes, is that because you're very skilled at refactoring?

Ideas for after the Dojo

- If you did this as a normal kata, try it as a refactoring kata (code on github[4])
- If you've done one of the three refactoring katas, try the other two. Were they easier or harder?
- Try doing all your refactoring without running the tests until you're "finished". How many tests did you break via refactoring mistakes?

[2]https://github.com/emilybache/Refactoring-Katas/tree/master/Tennis

[3]See the chapter (#ToolsForTheDojo)

[4]https://github.com/emilybache/Refactoring-Katas/tree/master/Tennis

Contexts to use this Kata

This is a good kata for practicing refactoring. There aren't many situations where you have the luxury of exhaustive tests. The three refactoring variants have slightly different challenges. The first two are by junior coders with poor grasp of the language. The third is designed to be as concise as possible, to the point of unreadability.