Mastering Coding Bat (Java)

Books by Ulm Publishing

Mastering Coding Bat (Java), Vol. 1: Basics
Mastering Coding Bat (Java), Vol. 2: Intermediate A
Mastering Coding Bat (Java), Vol. 3: Intermediate B
Mastering Coding Bat (Java), Vol. 4: Advanced

Mastering Coding Bat (Java)

Vol. 1: Basics

Gregor Ulm

Ulm Publishing

Copyright © 2018 by Gregor Ulm
http://www.gregorulm.com

All rights reserved. No part of this book may be reproduced in any form or by any means without the prior written consent of the copyright holder.

Contents

Preface

In 2012, I published what was back then the first full set of solutions to Coding Bat (codingbat.com), both in Java and Python, on my website (gregorulm.com). To my great surprise, there has been substantial interest, judging from the number of visitors to my website, and the deluge of comments and emails I have received over the years. The solutions I developed and published online I originally wrote down very quickly, making sure they look clean enough and pass all test cases.

Over time, I noticed that people started to view my solutions as a work of reference, comparing their solutions to mine. That was not my intention at all. However, as I am convinced of the didactic value of Coding Bat, I would like to further expand on my online solutions in a series of books. The goal of these books is to introduce problems, describe how to tackle them, give hints and, which is of utmost interest to people learning how to program, present fully-worked solutions that discuss how you get from an idea to a working piece of code. That happens in a number of steps, which may include discussing alternative approaches. By doing so, a novice programmer will make two important realizations. The first one is that there are many valid solutions to a given problem, while the second one is that non-trivial programming tasks are performed incrementally, which may involve dismissing earlier attempts.

The expected reader is a novice programmer, probably a freshman in college. You may also be a high school student or a self-learner. By working through the books of this series, you will gain a very thorough understanding of elementary programming concepts. This will provide an excellent preparation for future projects, either during your university studies or at work.

If you work through those problems, you will be well on your way to becoming a competent programmer. Seeing that programming skills have become important in many industries, that time will be well-spent.

I found it very gratifying to witness the positive reception my Coding Bat solutions have found. I hope that my book series *Mastering Coding Bat* will find an equally warm reception.

Gregor Ulm
Gothenburg, Sweden

Introduction

The exercise set published on the Coding Bat website is an excellent tool for leaning how to program. According to its creator, Stanford professor Nick Parlante, the motivation behind that site is to gradually introduce people to solving increasingly difficult problems programmatically. This is a particularly useful tool for novices because the gap from not knowing much to having to solve the kind of problems you are confronted with in an introductory course in computer science can be quite intimidating.

Programming has become a highly useful skill. Even if you have no aspirations of becoming a full-time software developer, it is not at all implausible that your field, no matter what it is, can advance dramatically with increased automation. For the most part, that is what programmers do: they automate processes. Imagine how much more effective you could be at your job if you were able to automate or outsource everything that is tedious and repetitive! If you fully understand all the problems on Coding Bat, you are well on the way of joining the automators, if you so desire.

With this book, the goal is to dissect every problem in the four Coding Bat sections *Warmup-1*, *String-1*, *Array-1*, and *Logic-1*. Those sections introduce basic operations, which are the bread and butter of programming. They gradually introduce new concepts, but not too many of them. There is also a significant amount of problems that are variations of previous ones, which allow you to solidify your knowledge.

This book is primarily for everyone who has encountered Coding Bat and would like to get a bit more hand-holding, better guid-

ance, or some help with understanding the various problems. You could be a high school student, or maybe you are enrolled in college. You may even be an autodidact. No matter your role, you may have encountered the problem that even books and other materials that are supposedly written for beginners assume a significant amount of knowledge. Even worse, many of those books are incredibly, incredibly verbose. In contrast, this book presents you with the meat of programming. If you make it through it, you will become a better programmer.

How to use this book

View this book as your personal tutor who guides you through Coding Bat. The presentation of each problem has a fixed format. We start with the *problem description*, which also contains a skeleton of the code, i.e. a function signature. It is your task to come up with the body of the function. There are many ways to skin a cat, and there are at least as many ways to solving problems programmatically. However, some approaches keep you from learning certain skills. Thus, this book lists the *tools* you can use for solving a given problem. It may be tempting to just use an in-built method in Java that does what you are supposed to write code for, but if you do that, you are not going to progress much.

In order to guide you in the right directions, the *hints* section provides a few pointers. Depending on the problem, those can be substantial or almost nonexistent. Sometimes it is just a reference to a previous and similar problem. This is followed by the normally only sparsely commented *solution*, which shows one possible solution to the problem. It will be correct, but it may not be the most elegant one. In contrast, the optional *discussion* section highlights different approaches or presents variations. Some problems come with a very extensive discussion section that contains several alternative solutions.

Treat this book essentially as very extensive commentary. I would suggest that you work through it sequentially as the problems tend to increase in difficulty. If you jump around and get stuck, you may only get needlessly frustrated. For each problem, start by reading the problem description. Then look at the code skeleton. Instead of writing code on the Coding Bat website, however, I would like you to use pencil and paper. Yes, you read

that correctly. Before starting to type, write code by hand and try to come up with a solution that is as close to working code as possible. By not writing executable Java code straight away you will not get tempted to, for instance, guessing and using the website as some kind of feedback generator that helps you to gradually figure out the solution. Instead, with your pencil in hand, you figure out the problem before writing any code on the computer. If you cannot do that, then you may need to think harder about the problem. As you get more experienced, you may want to skip writing code down on paper first. However, if you find yourself changing your code too much, then you probably need to spend more time figuring out the solution in your head first.

Once you are content with the code you have written down on paper, go to the Coding Bat website. Enter your solution and execute the code. Fix syntactic errors if there are any. (Optional: if you get stuck, use my hints.) After you have solved the problem correctly, look at my solution and the discussion. Compare your solution with mine. Make sure you understand every single line in both your code and my solutions as well as all the variations.

Lastly, I would recommend you work steadily through this book. Do at least one or two problems every day, but ideally more. I think you should be able to do at least five problems a day very comfortably.

Warmup-1

notString

Problem

In the problem `notString`, we encounter strings. The function takes a string `str` as an argument and adds `"not "` to it, including a space at the end. However, this only happens if the string `str` does not start with `"not"`. If it does start with `"not"`, then return `str` unchanged. See Listing 49 for the code skeleton.

```
1 public String notString(String str) {
2 // your code here
3 }
```

Listing 49: `notString` – skeleton

Tools

You can use conditional statements, comparison operators, boolean operators, the string concatenation operator as well as the string methods `length`, `substring`, and `equals`.

Hints

The problem description does not indicate that there is an issue with the method `substring`: If the string you are applying this method to is not within the boundaries of the substring you have specifed, Java throws an exception, i.e. your program crashes. Consequently, you first need to check the length of the string. In case the string `str` is of the correct length, check if the first three characters of it equal `"not"`.

Solution

Based on the hints provided above, one possible solution is given in Listing 50. We systematically perform the required checks. If any of the checks fail, we know that we need to prepend the string with `"not "`. Obviously, if the string is not at least three characters long, it cannot start with `"not"`. Therefore, we can return a new string, consisting of the input string `str`, prepended with `" not"`. We also do this if `str` is at least three characters long, but does not start with `"not"`. Otherwise, we return `str`.

```
1 public String notString(String str) {
2   if (str.length() >= 3) {
3     String prefix = str.substring(0, 3);
4     if (prefix.equals("not")) {
5       return str;
6     }
7   }
8   return "not " + str;
9 }
```

Listing 50: `notString` – solution

Discussion

Let us start by tackling this problem case by case. The first important observation is that if the input string `str` is less than three characters long, we have to prepend it with `"not "`. After all, we are checking if `str` starts with `"not"`, and if it is not at least three characters long, it is not possible that it could start with that word. If the function `notString` has not returned after the first if-clause, we perform a second check, based on the first three letters of the string. If that substring equals `"not"`, we return the string `str` unchanged. Finally, if the function still has not returned, we return a new string, made of `str` and a prepended string `"not "`. The corresponding code is provided in Listing 51. We will look at a few more variations further below.

```
1 public String notString(String str) {
2   if (str.length() < 3) {
3     return "not " + str;
4   }
5   if (str.substring(0, 3).equals("not")) {
6     return str;
7   }
8   return "not " + str;
9 }
```

Listing 51: `notString` – variation 1

In order to further simplify the code, it would be helpful if the first if-clause could be modified so that it returns `str`. For this, we need to combine the two if-clauses, for example by nesting, as shown in Listing 52.

```
1 public String notString(String str) {
2   if (str.length() >= 3) {
3     if (str.substring(0, 3).equals("not")) {
4       return str;
5     }
6   }
7   return "not " + str;
8 }
```

Listing 52: `notString` – variation 2

We can condense the code further by removing the nested if-clause, as shown in Listing 53. I would not recommend turning the resulting code into a one-liner by using the ternary operator as the resulting code would be difficult to read.

```
1 public String notString(String str) {
2   if (str.length() >= 3
3       && str.substring(0, 3).equals("not")) {
4     return str;
5   }
6   return "not " + str;
7 }
```

Listing 53: `notString` – variation 3

in1020

Problem

The function `in1020` takes two integers `a`, `b` as input and returns `true` if one of them is in the range [10 . . . 20]. Otherwise, it returns `false`. See Listing 87 for the code skeleton.

```
1 public boolean in1020(int a, int b) {
2 // your code here
3 }
```

Listing 87: `in1020` – skeleton

Tools

For this problem you can use conditional statements, comparison operators, and boolean operators.

Hints

This is very straightforward. As a start, tackle both cases in turn, i.e. you first check `a`, followed by `b` and determine if they are in the prescribed range of integers.

Solution

The code in Listing 88 shows a straightforward solution. In the subsequent discussion, we will see a more elegant approach, though.

```
public boolean in1020(int a, int b) {
  return (a >= 10 && a <= 20) || (b >= 10 && b <= 20);
}
```

Listing 88: `in1020` – solution

Discussion

Let us start by checking both integers in separate if-statements. As you can see in the code shown in Listing 89, there is quite some duplication. We perform the exact same check, on different values, twice. This does not change if we rewrite the code into a more concise form, like in Listing 88 above.

```
public boolean in1020(int a, int b) {
  if (a >= 10 && a <= 20) {
    return true;
  }
  if (b >= 10 && b <= 20) {
    return true;
  }
  return false;
}
```

Listing 89: `in1020` – variation 1

Indeed, a downside of this exercise is that it teaches you to duplicate code, which is a rather bad habit to acquire. If you encountered a problem like this in the real world, you would instead create a separate helper function. This is goes a bit beyond what Coding Bat teaches up to this point, so view the code given in Listing 90 as a reference. As you can see, the helper function `inRange` checks whether the argument `n` is within the desired range. Thus, we replace the actual computation shown in Listing 88 with two function calls.

```
public boolean in1020(int a, int b) {
  return inRange(a) || inRange(b);
}

public boolean inRange(int n) {
  return n >= 10 && n <= 20;
}
```

Listing 90: in1020 – variation 2