Exercise #4.1 IF Flow Control

If-else statement design can be problematic if boundary conditions are not defined properly. The property of code can move from general-logic-based to specific-case-defined. Even though later fixed up can cover some cases failed before, the quality of code is probably still weak and vulnerable to extreme case (related to quality of test cases).

Problem Description:

In Black Jack card game, we call a value *busts* if it exceeds a sum of 21. Write a Java program *BlackJack* that on input 2 integer values greater than 0, prints a single integer of whichever value is nearest to 21 without going bust. Print -1 if both values bust.

Key information:

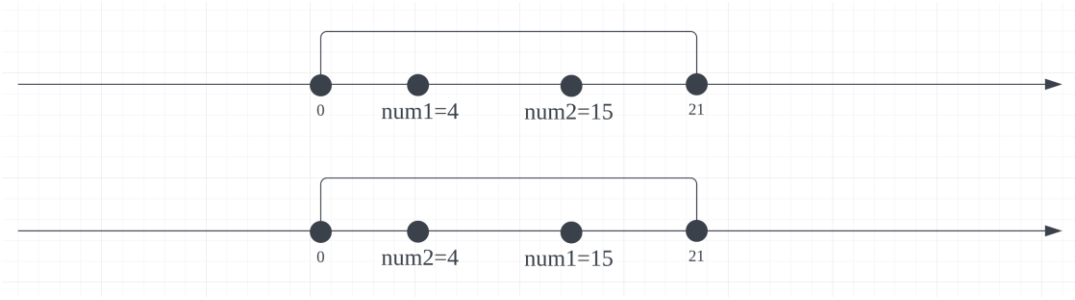
Two integer input greater than 0; valid interval is [1,21]

Solution Design

We use one axis to demonstrate the possible scenarios in the above problem.

valid interval is [1,21]. The range of input are identical---[1,∞).

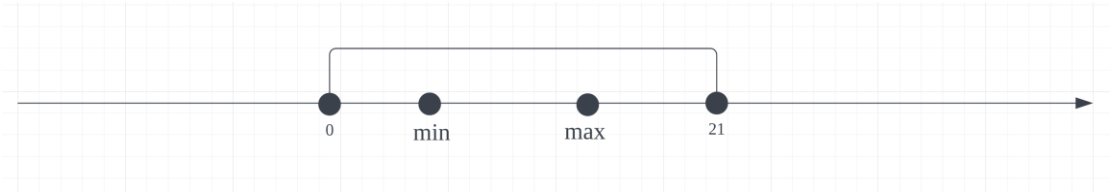
However, the relation between two input numbers---num1 and num2 is unknown. So we might have two equivalent scenarios where num1 and num2 switch their value.



To avoid considering equivalent scenarios again, we can solve their numerical relation before we move on. Now we define minNum=min(num1,num2) and maxNum=max(num1,num2).

Then we have following scenarios

- Scenario 1:valid



- Scenario 2: valid

- Scenario 3: not valid

Next:

Now we have worked out all three scenarios and we can write Boolean conditions for each scenario.

- For Scenario 1:

min<=max<=21, then print max

Note that we leave out min>0 because the description of question says the input are

positive integers.

- For Scenario 2:

min<=21<=max, then print min

- For Scenario 3:

21<=min<=max, then print -1

Coding:

Now we can use three "If" statements to print correct answers for each scenario.

implementation in java files in "src" folder

Optimization of design:

Three consecutive "IF" statements are sufficient for this problem but this coding strategy still has limitations. Since a java program runs line-by-line consecutively, in the above implantation, even if inputs lie in the first scenario, the java program will execute the "if" statement and check whether the inner condition is True. Thus, these "if" statements could waste a lot of computation resources (especially when we have thousands of scenarios).

However, the "if-else if-…-else" block can optimize the code. In "if-else if-…-else" statements, once a condition "x" of any "If" or "else if" is true, then the program will skip later parts of "if-else if-…-else" blocks after it finishes the first code block behind condition "x".

Check comments of java implementation for optimized code

Harder Version:

The inputs are no longer positive integer but two integers.

Follow the design strategy and try to solve the harder version. Solution in this repo. Enjoy yourself.