Introduction

Participant Information Sheet

Project Title: Impact of Code Decisions on Code Understanding

Research Team: Assoc Prof Ewan Tempero (Principal investigator), Prof Andrew Luxton-Reilly (Co-investigator), Assoc Prof Paul Denny (Co-investigator), Dr Yu-Cheng Tu (Co-investigator), Dr Burkhard Wuensche (Co-investigator)

My name is Ewan Tempero and I would like to invite you to take part in a research study. Before you decide whether to participate, you need to understand why the research is being done and what it would involve for you. Please take time to read the following information carefully. Ask questions if anything you read is not clear or if you would like more information. Take time to decide whether or not you wish to take part.

The Study

Programs that perform the same task can be written in many different ways using different elements of the language (similarly to instructions given in English). It is widely believed that the choice of programming elements, and the arrangements of those elements, impacts the comprehensibility of the code. In other words, some programs will be easier to understand than other programs that perform the same task. We aim to compare the comprehensibility of different ways of writing programs to determine which approaches are easier to understand. This has implications for both teaching and industry practice because we can recommend that authors use forms of programming code that is easier to understand.

What Will You Have to do?

The survey will be conducted online and anonymously. By clicking on the “Start survey” link, you are providing your consent for your data from this survey to be collected and studied. All data will be stored anonymously, so there will be no way to connect this data with you. The survey will be deployed using Qualtrics and the anonymous responses to the survey will be stored indefinitely. If you are a student of the researchers, we give our assurance that your participation or non-participation in this study will have no effect on your grades or relationship with the University and that you may contact your academic head should you feel that this assurance has not been met.
This is expected to take approximately 30 minutes.
By participating in this study, you will help the researchers to understand and compare the comprehensibility of different elements of programming code.

Why Have You Been Invited to Take Part?

You have been invited to take part in the study as a student with some programming experience.

Confidentiality

Confidentiality will be maintained throughout the study as all responses are anonymous.

How Will the Information Gathered in this Study be Stored and Protected?

Any information gathered during this study will be saved securely on University of Auckland file systems before being archived indefinitely in a public archive accessible through the Figshare research data-sharing platform. All responses to the questionnaire are anonymous, so you will not be able to withdraw your data once it has been provided.
If you wish to receive a summary of the results then you may provide your email address at the end of the study. Your email address will not be linked to the questionnaire data. The email address will be stored securely on University of Auckland file systems until the study is complete and a summary of results has been distributed, and then it will be deleted.

What Will Happen to the Results of this Study?

The results may be published in academic venues such as journals or conferences or in academic presentations.

Who Should you Contact for Further Information?

Ewan Tempero (Principal investigator) 09-923-3765, e.tempero@auckland.ac.nz
Giovanni Russello (Head of School), 09-923-6137, g.russello@auckland.ac.nz

UAHPEC Chair contact details:

For any queries regarding ethical concerns you may contact the Chair, The University of Auckland Human Participants Ethics Committee, Office of Research Strategy and Integrity, The University of Auckland, Private Bag 92019, Auckland 1142. Telephone 09 373-7599 ext. 83711. Email: humanethics@auckland.ac.nz

Approved by the University of Auckland Human Participants Ethics Committee on 3 March 2023 for three years. Reference Number UAHPEC25663.

Please note: By clicking "Agree" below, you consent to have your anonymous responses to this questionnaire recorded and used in our study. Submission of the questionnaire will be used as consent to take part in the research.

The survey should take about 30 minutes to complete on average.

○ Agree

○ Disagree

Instructions

In the following pages you will be presented with the following sets of questions:

- Demographics - 4 questions to describe your programming experience.
- Pretest - 4 questions testing your understanding of basic Java.
- Main questionnaire - A sequence of 2 pieces of Java code, and questions about each piece.
- Final comments.

Request for results - you will have an opportunity to provide your email address if you would like to see the results. Your email address will not be associated with your responses to this study.

The usefulness of your responses depends on them coming just from you, so please complete the questions without the aid of other people, use of information from websites or from other external sources. Also, do not use other tools or devices to help you answer the questions.

Demographics Block

What is the most advanced course you have completed or are currently enrolled in?

- ○ 100-level COMPSCI
- ○ 200-level COMPSCI
- ○ 300-level COMPSCI
- ○ 700-level COMPSCI
- ○ 200-level SOFTENG
- ○ 300-level SOFTENG
- ○ 700-level SOFTENG
- ○ Other 100-level programming course
- ○ Other 200-level programming course
- ○ Other 300-level programming course
- ○ Other 700-level programming course
- ○ Other programming course
- ○ No course

How would you describe your programming experience in any programming language? Check all that apply.

- ☐ University courses only
- ☐ Non university courses (formal exercises, e.g. high school, on-line courses)
- ☐ Self taught (writing code not just for course exercises)
- ☐ Significant own projects (beyond just learning a language)
- ☐ Industry experience
- ☐ Open-source contributor
- ☐ Other (please specify) ______

How would you rate your **experience** in programming with Java

- ○ None
- ○ Written tiny programs (up to 100 lines each)
- ○ Written small programs (100-500 lines each)
- ○ Written medium programs (500-2000 lines)
- ○ Written programs larger than 2000 lines

How would you describe your **confidence** in programming in Java

- ○ Very low (unable to complete simple programs without help)
- ○ Low (able to complete simple programs but not confident to do large programs, or there are several features of Java that I am not confident with)
- ○ Medium (able to complete non-trivial programs, but not familiar with all aspects of Java)
- ○ High (able to complete large programs, there may be some rarely-used unfamiliar aspects of Java)
- ○ Very high (able to complete any Java related programming)

PreTest Block

The following questions test some aspects of Java that will be used later in the questionnaire. Please be as accurate as possible, but if you are unsure then choose that option.

```
String[] list = {"two", "words"};
System.out.println(list.length);
```

What value is printed when the code above is executed?

- ○ 0
- ○ 1
- [✓ ○ 2]
- ○ 3
- ○ A NullPointerException will be thrown
- ○ Unsure

```
int[] list = { 10, 9, 8, 7, 6 };
for (int elt: list) {
    System.out.println(elt);
}
```

What value is printed when the code above is executed?

○ 10

○ 6

[✓ ○ 10]
```
9
8
7
6
```

○ 6
```
7
8
9
10
```

○ 9
```
8
7
```

○ Unsure

```
int[] list = { 1, -1, 2, -2, 3, -3 };
int count = 0;
for (int value: list) {
    if (value > 0) {
        count = count + 1;
    }
}
System.out.println(count);
```

What value is printed when the code above is executed?

- 0
- 1
- 2
- [✓ 3]
- 4
- Unsure

```
int a = 10;
int b = 20;
boolean c = a > b;
System.out.println(c);
```

What value is printed when the code above is executed?

- 0
- 1
- true
- [✓ false]
- a > b
- Unsure

```
int[] inputData = { 1, -1, 2, -2, 3, -3 };
List<Integer> list = new ArrayList<Integer>();
for (int value: inputData) {
    if (value >= 0) {
        list.add(value);
    }
}
System.out.println(list);
```

What value is printed when the code above is executed?

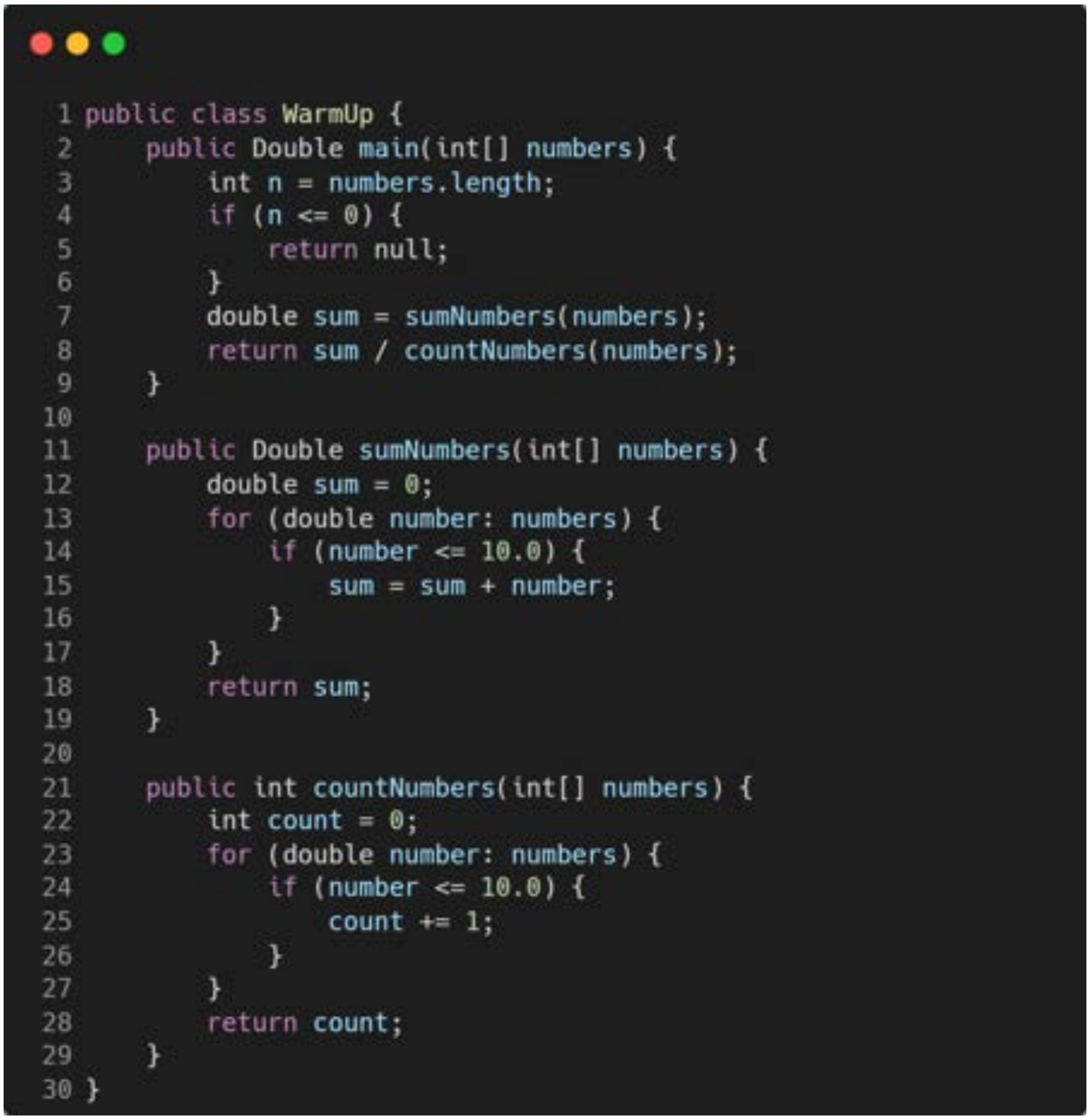
- (nothing printed)
- [-3]
- [✓ [1, 2, 3]]
- [-1, -2, -3]
- [1, -1, 2, -2, 3, -3]
- Unsure

Progress Info

The main questionnaire will begin in the following pages where you will see a sequence of 2 pieces of Java code, and questions about each piece. Click next to continue.

Warm up

Instructions: Study the code presented below. You will be asked a question about this code on the following page. Once you leave this page you will not be able to return, so spend as much time as you need to understand this code.
You may assume there are no errors in the code.



```java
public class WarmUp {
    public Double main(int[] numbers) {
        int n = numbers.length;
        if (n <= 0) {
            return null;
        }
        double sum = sumNumbers(numbers);
        return sum / countNumbers(numbers);
    }

    public Double sumNumbers(int[] numbers) {
        double sum = 0;
        for (double number: numbers) {
            if (number <= 10.0) {
                sum = sum + number;
            }
        }
        return sum;
    }

    public int countNumbers(int[] numbers) {
        int count = 0;
        for (double number: numbers) {
            if (number <= 10.0) {
                count += 1;
            }
        }
        return count;
    }
}
```

Write 2-3 sentences explaining what the code you have just studied on the previous page does. Your description should be at a high-level but still be complete.

Rate your confidence of the accuracy of your answer.

- ○ Low
- ○ Medium
- ○ High

Here is our answer to illustrate the kind of answer we are looking for:

Computes the average of all the numbers in the list that are no larger than 10.

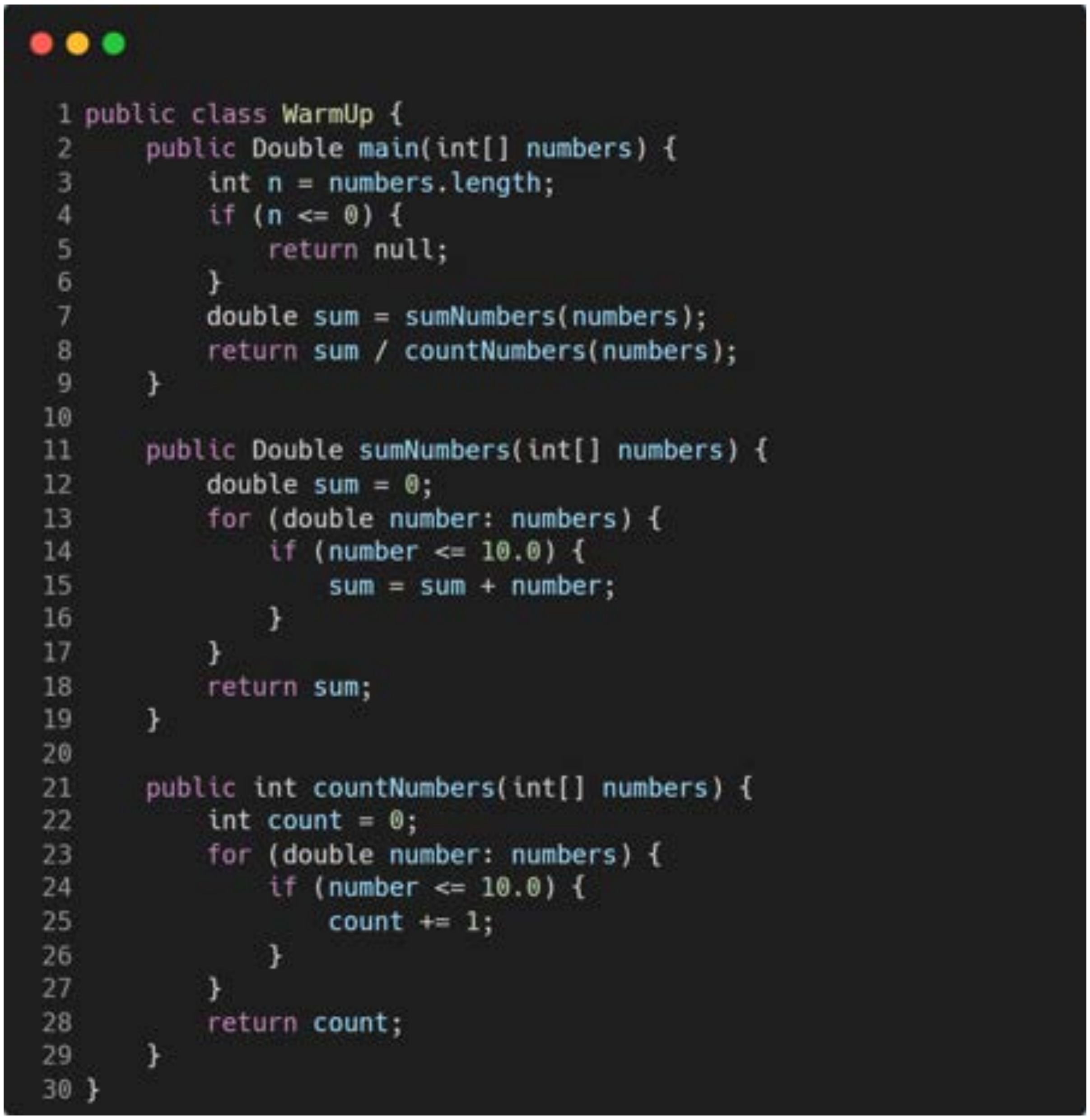
Notes:

- An example of an incomplete answer is "Computes the average of a list". This does not mention that not all values in the list are used, nor why some numbers are not used.
- An example of a answer that is too low-level is "Runs through a list of numbers summing those no larger than 10 and then dividing by the number summed."

Instructions: Below is the same code again. Answer the following question about it.

If the input `[ 1.0, 2.0, 3.0 ]` is passed to the function `main()`, what is the output?

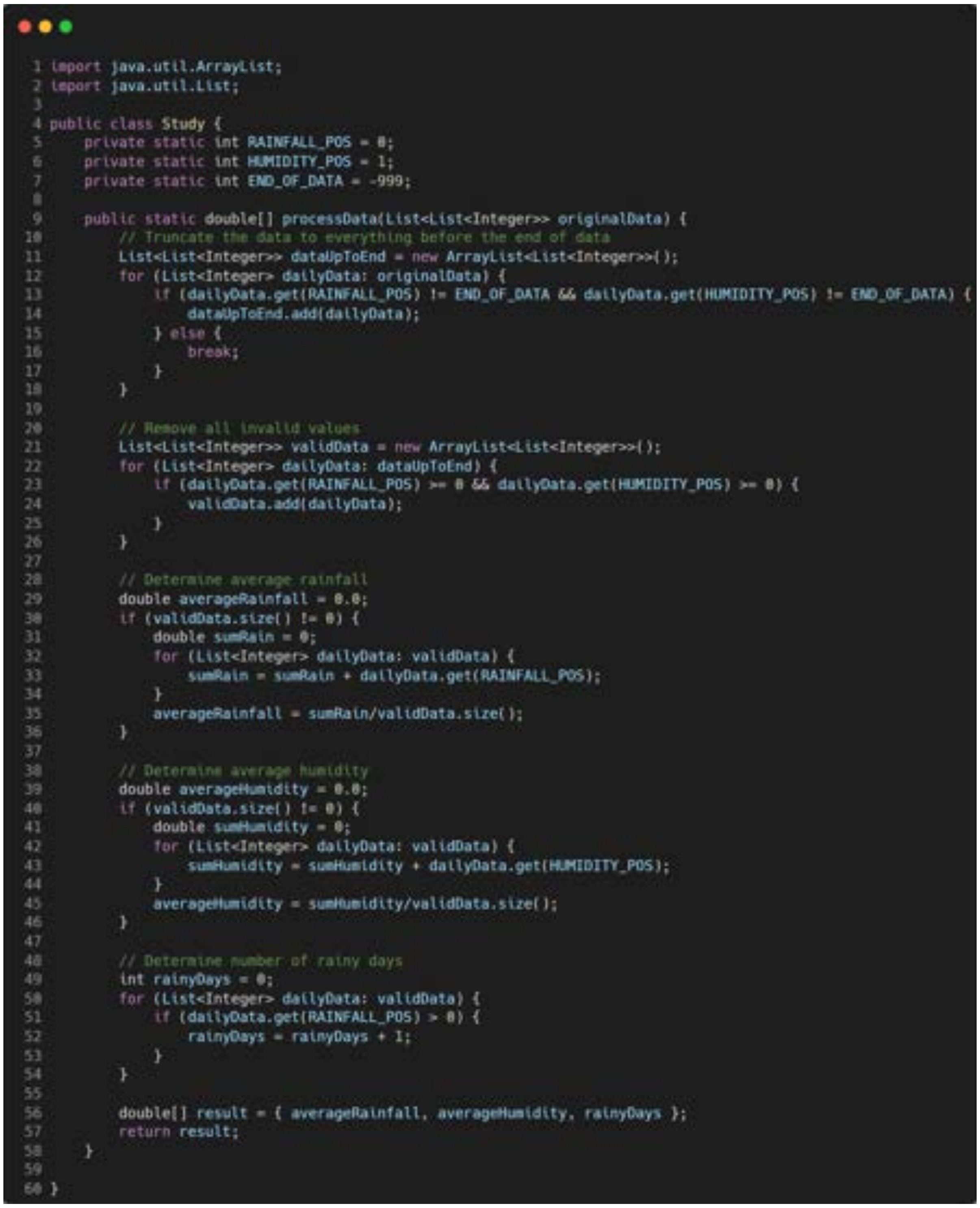
- ○ 1.0
- [✓ ○ 2.0]
- ○ 3.0
- ○ [1.0, 2.0, 3.0]
- ○ None
- ○ Unsure



```java
public class WarmUp {
    public Double main(int[] numbers) {
        int n = numbers.length;
        if (n <= 0) {
            return null;
        }
        double sum = sumNumbers(numbers);
        return sum / countNumbers(numbers);
    }

    public Double sumNumbers(int[] numbers) {
        double sum = 0;
        for (double number: numbers) {
            if (number <= 10.0) {
                sum = sum + number;
            }
        }
        return sum;
    }

    public int countNumbers(int[] numbers) {
        int count = 0;
        for (double number: numbers) {
            if (number <= 10.0) {
                count += 1;
            }
        }
        return count;
    }
}
```

Weather SFV

Instructions: Study the code presented below. You will be asked a question about this code on the following page. Once you leave this page you will not be able to return, so spend as much time as you need to understand this code.
You may assume there are no errors in the code.



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

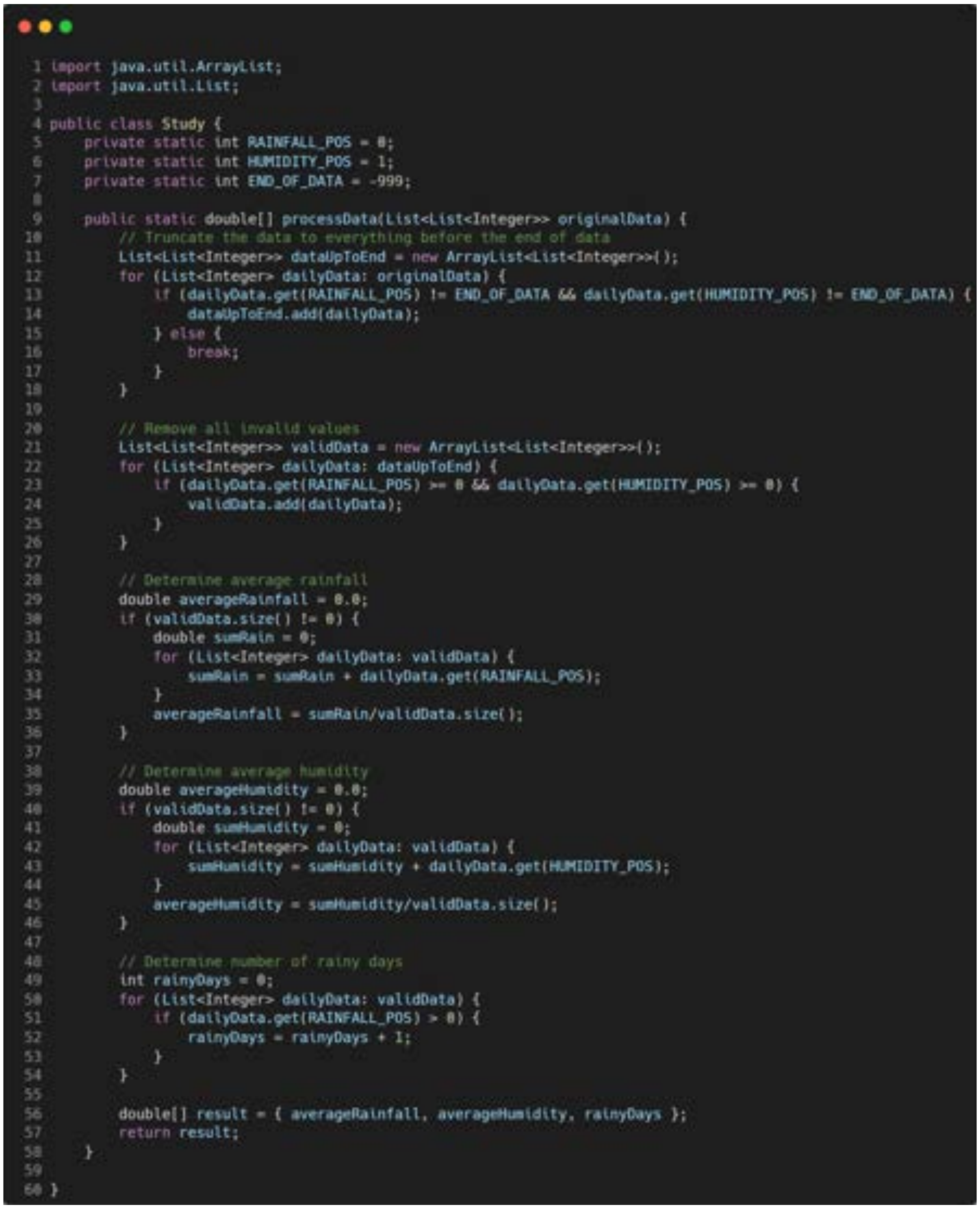
Write 2-3 sentences explaining what the code you have just studied on the previous page does. Your description should be at a high-level but still be complete.

Rate your confidence of the accuracy of your answer.

- Low
- Medium
- High

Q1 Given the same code below. If the input `[[1,1], [1,1], [1,1]] (list of 3 elements, each element a list of 2 values)` is passed to the function `main()`, what is the output?

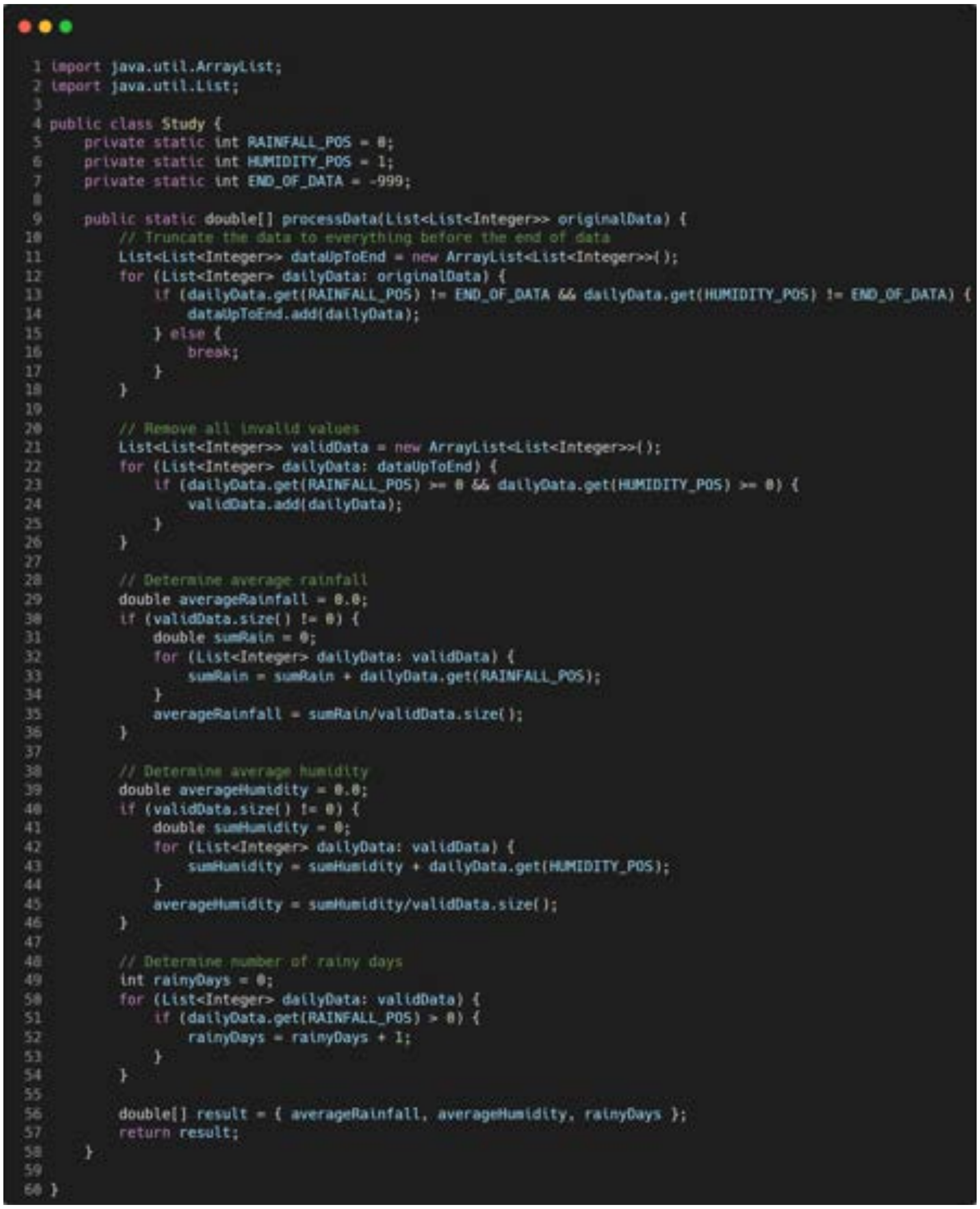
- [1, 1, 1]
- [3, 3, 3]
- [3, 1, 1]
- [1, 3, 1]
- [✓ [1, 1, 3]]



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Q2 Given the same code below. If the input `[[-3,-3], [-3,-3]]` is passed to the function `main()`, what is the output?

- ○ [-3, -3, -3]
- ○ [-3, -3, 2]
- [✓ [0, 0, 0]]
- ○ [0, 0, 2]
- ○ [-3, -3, 0]

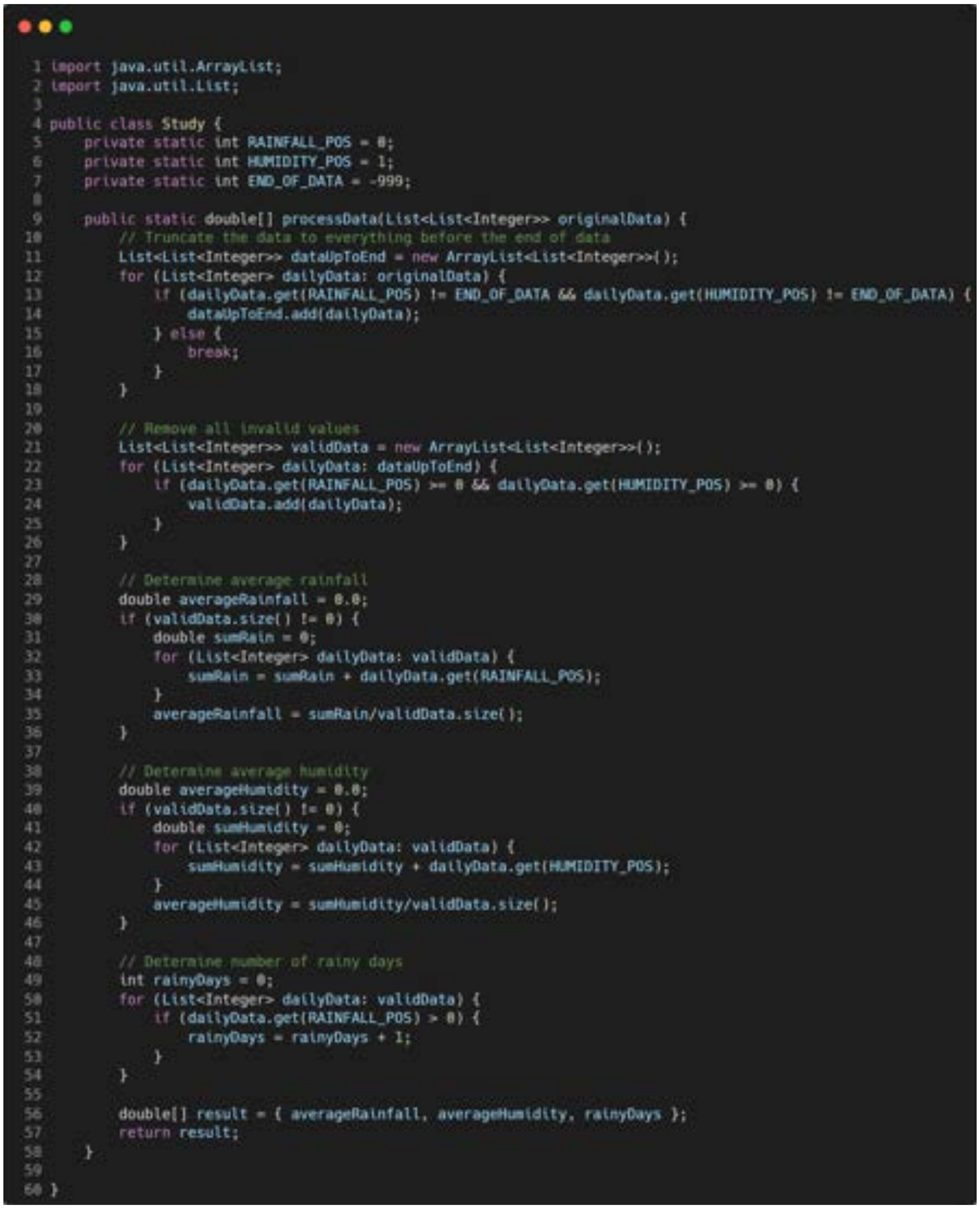


```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Q3

Given the same code below. If the input `[[1,1], [1,1], [-999,-999], [1,1]]` is passed to the function `main()`, what is the output?

- [1, 1, -999]
- [2, 2, -999]
- [3, 3, -999]
- [0, 0, 0]
- [✓ [1, 1, 2]]

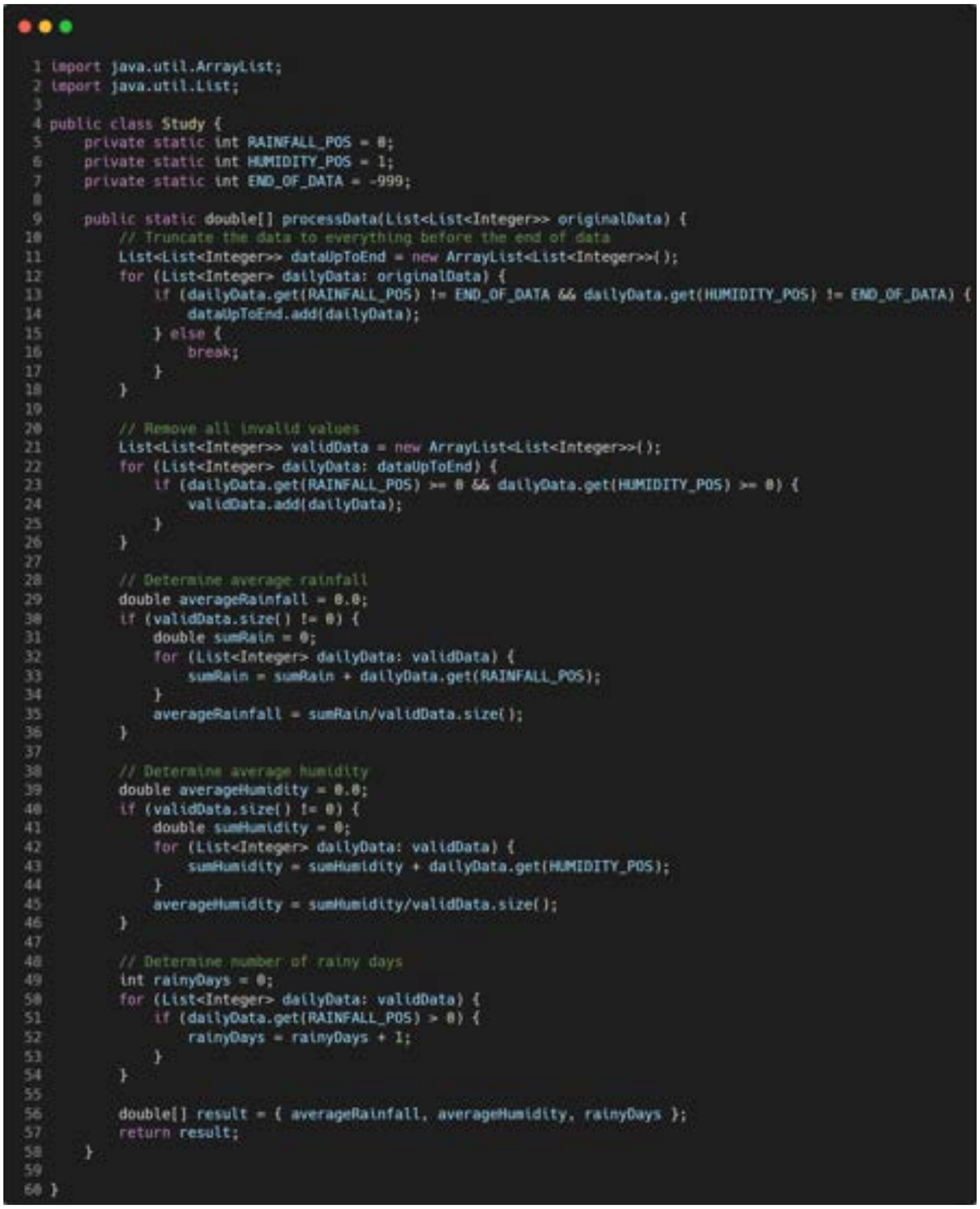


```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Q4

Given the same code below. What input to `main()` would produce the output `[0, 0, 0]`

- ○ [[0,0], [1,1], [1,2]]
- ○ [[0,0], [-1,-1], [1,2]]
- ○ [[-1,-1], [1,1], [1,2]]
- [✓ ○ [[0,-999], [0,0], [1,2]]]
- ○ [[0,0], [1,1], [1,-999]]

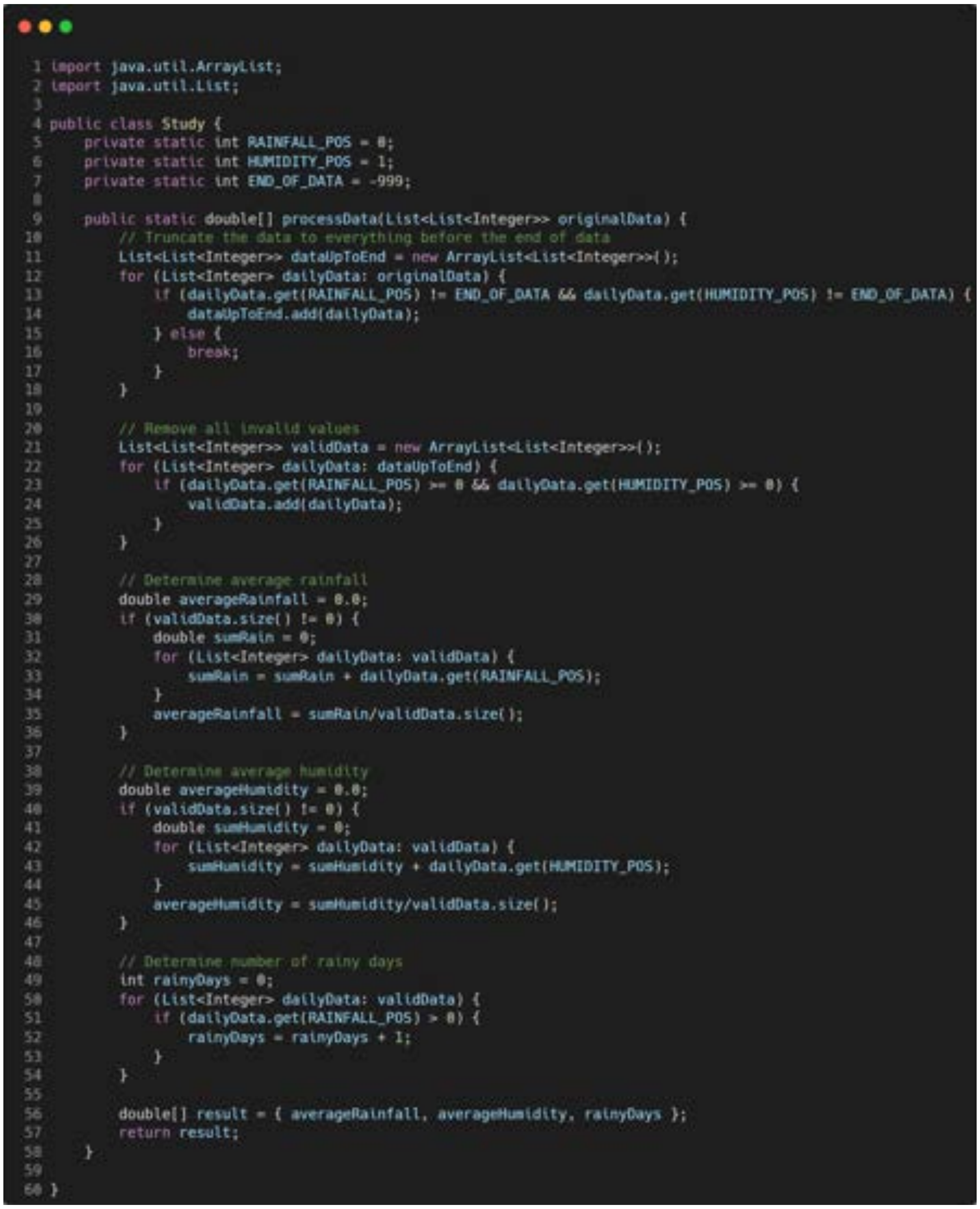


```
 1 import java.util.ArrayList;
 2 import java.util.List;
 3
 4 public class Study {
 5     private static int RAINFALL_POS = 0;
 6     private static int HUMIDITY_POS = 1;
 7     private static int END_OF_DATA = -999;
 8
 9     public static double[] processData(List<List<Integer>> originalData) {
10         // Truncate the data to everything before the end of data
11         List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
12         for (List<Integer> dailyData: originalData) {
13             if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
14                 dataUpToEnd.add(dailyData);
15             } else {
16                 break;
17             }
18         }
19
20         // Remove all invalid values
21         List<List<Integer>> validData = new ArrayList<List<Integer>>();
22         for (List<Integer> dailyData: dataUpToEnd) {
23             if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
24                 validData.add(dailyData);
25             }
26         }
27
28         // Determine average rainfall
29         double averageRainfall = 0.0;
30         if (validData.size() != 0) {
31             double sumRain = 0;
32             for (List<Integer> dailyData: validData) {
33                 sumRain = sumRain + dailyData.get(RAINFALL_POS);
34             }
35             averageRainfall = sumRain/validData.size();
36         }
37
38         // Determine average humidity
39         double averageHumidity = 0.0;
40         if (validData.size() != 0) {
41             double sumHumidity = 0;
42             for (List<Integer> dailyData: validData) {
43                 sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
44             }
45             averageHumidity = sumHumidity/validData.size();
46         }
47
48         // Determine number of rainy days
49         int rainyDays = 0;
50         for (List<Integer> dailyData: validData) {
51             if (dailyData.get(RAINFALL_POS) > 0) {
52                 rainyDays = rainyDays + 1;
53             }
54         }
55
56         double[] result = { averageRainfall, averageHumidity, rainyDays };
57         return result;
58     }
59
60 }
```

Q5

Given the same code below. Suppose you had to change to the code to ignore values larger than 100. Which of the following sets of lines would you need to modify (one or more lines) to achieve this change?

- 11-18
- [✓ 21-26]
- 29-36
- 39-46
- 49-54

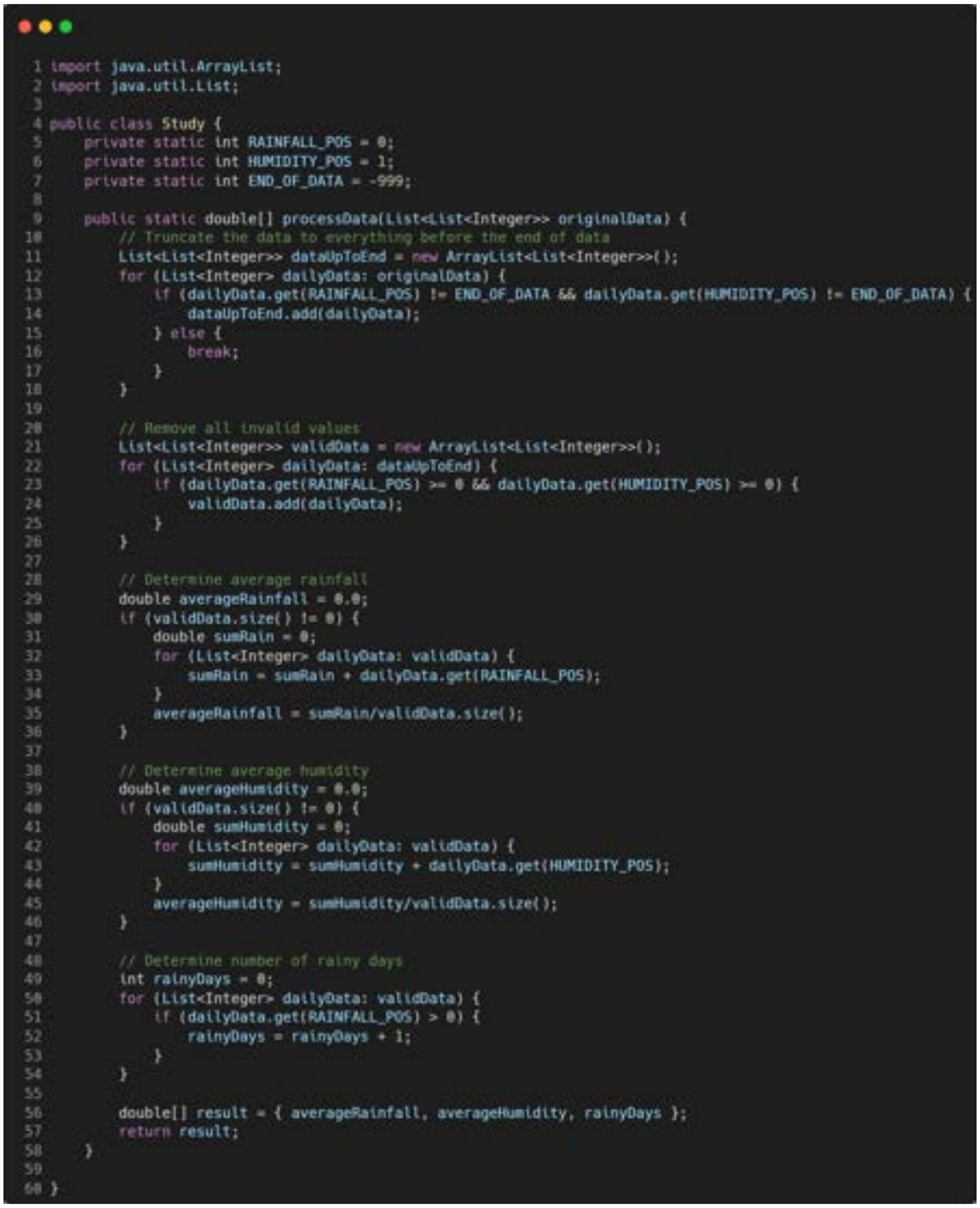


```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Given the same code below. How easy do you think this code is to understand?

- ○ Very easy
- ○ Somewhat easy
- ○ Not easy, but not difficult
- ○ Difficult
- ○ Very difficult

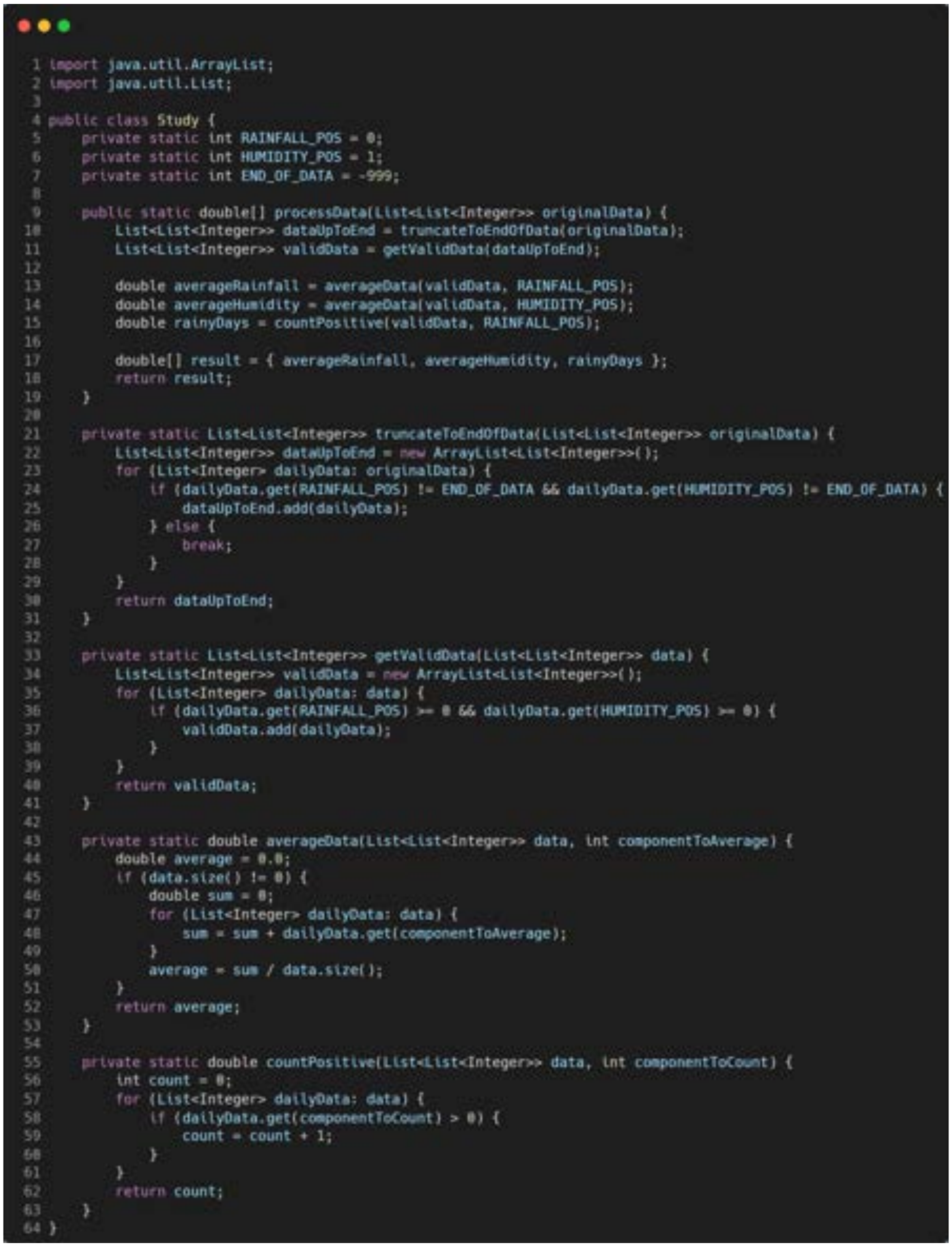
Comment on your response (optional)



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Weather MFV

Instructions: Study the code presented below. You will be asked a question about this code on the following page. Once you leave this page you will not be able to return, so spend as much time as you need to understand this code.
You may assume there are no errors in the code.



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

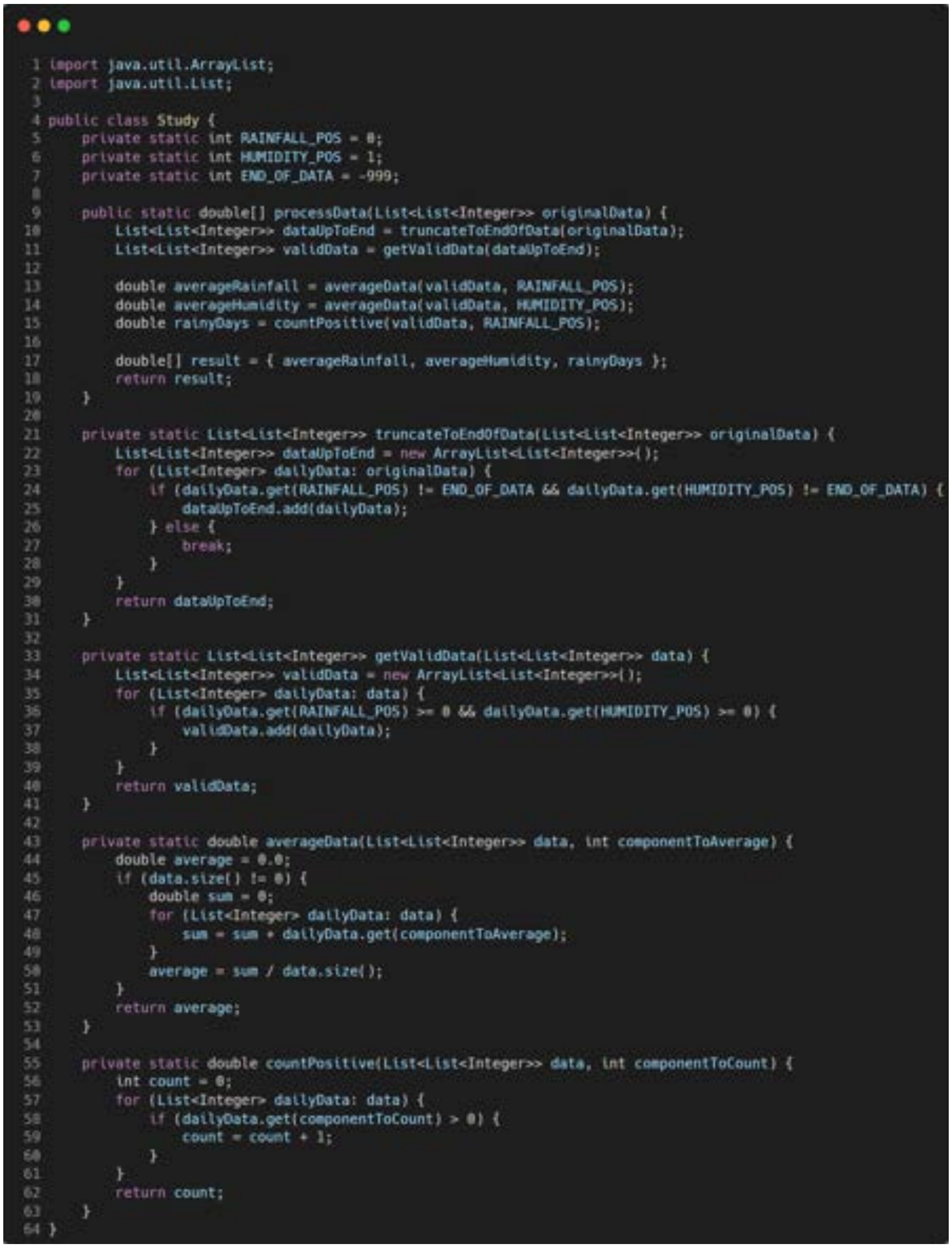
Write 2-3 sentences explaining what the code you have just studied on the previous page does. Your description should be at a high-level but still be complete.

Rate your confidence of the accuracy of your answer.

- ○ Low
- ○ Medium
- ○ High

Q1 Given the same code below. If the input `[[1,1], [1,1], [1,1]]` (list of 3 elements, each element a list of 2 values) is passed to the function `main()`, what is the output?

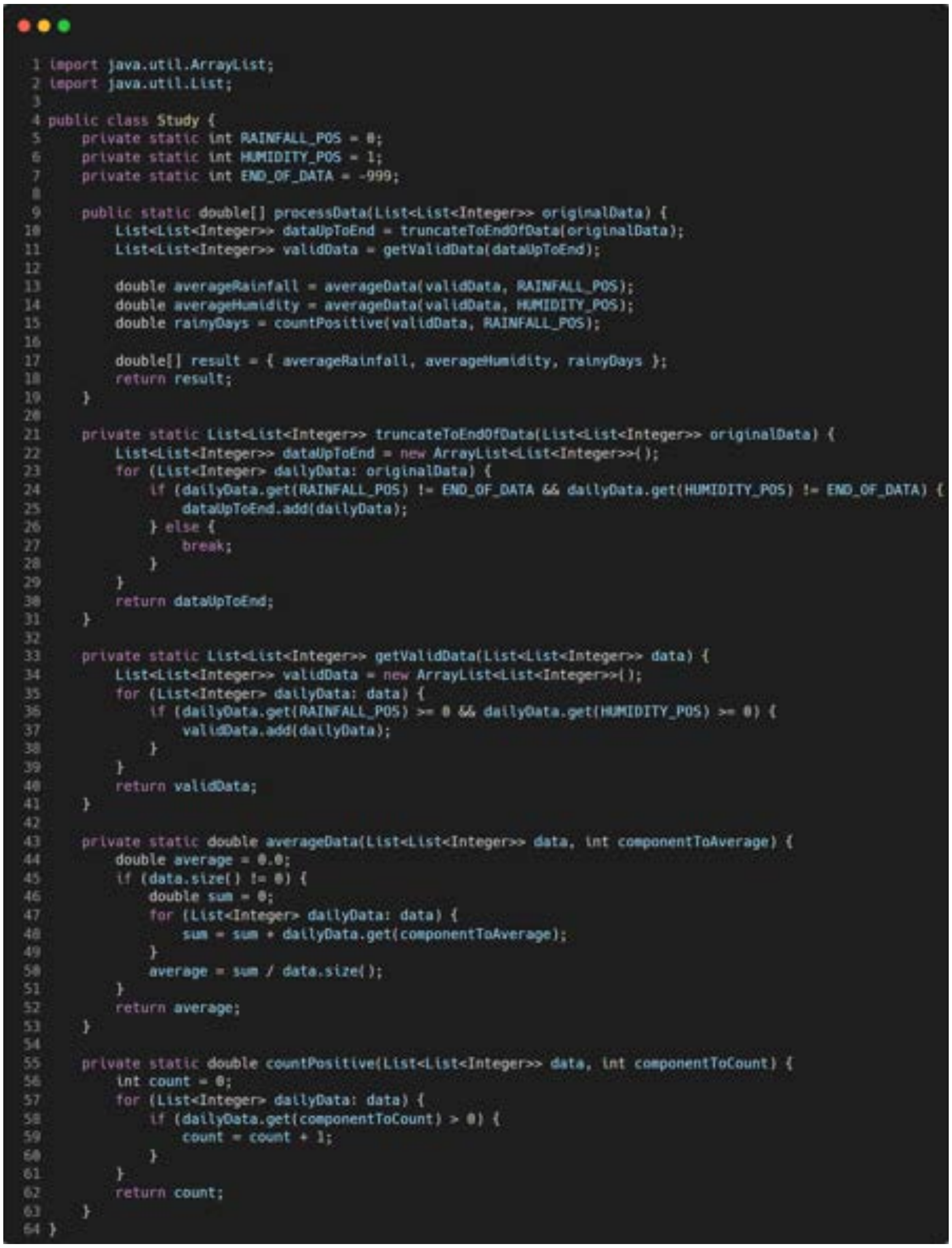
- ○ [1, 1, 1]
- ○ [3, 3, 3]
- ○ [3, 1, 1]
- ○ [1, 3, 1]
- [✓ ○ [1, 1, 3]]



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Q2 Given the same code below. If the input `[[-3,-3], [-3,-3]]` is passed to the function `main()`, what is the output?

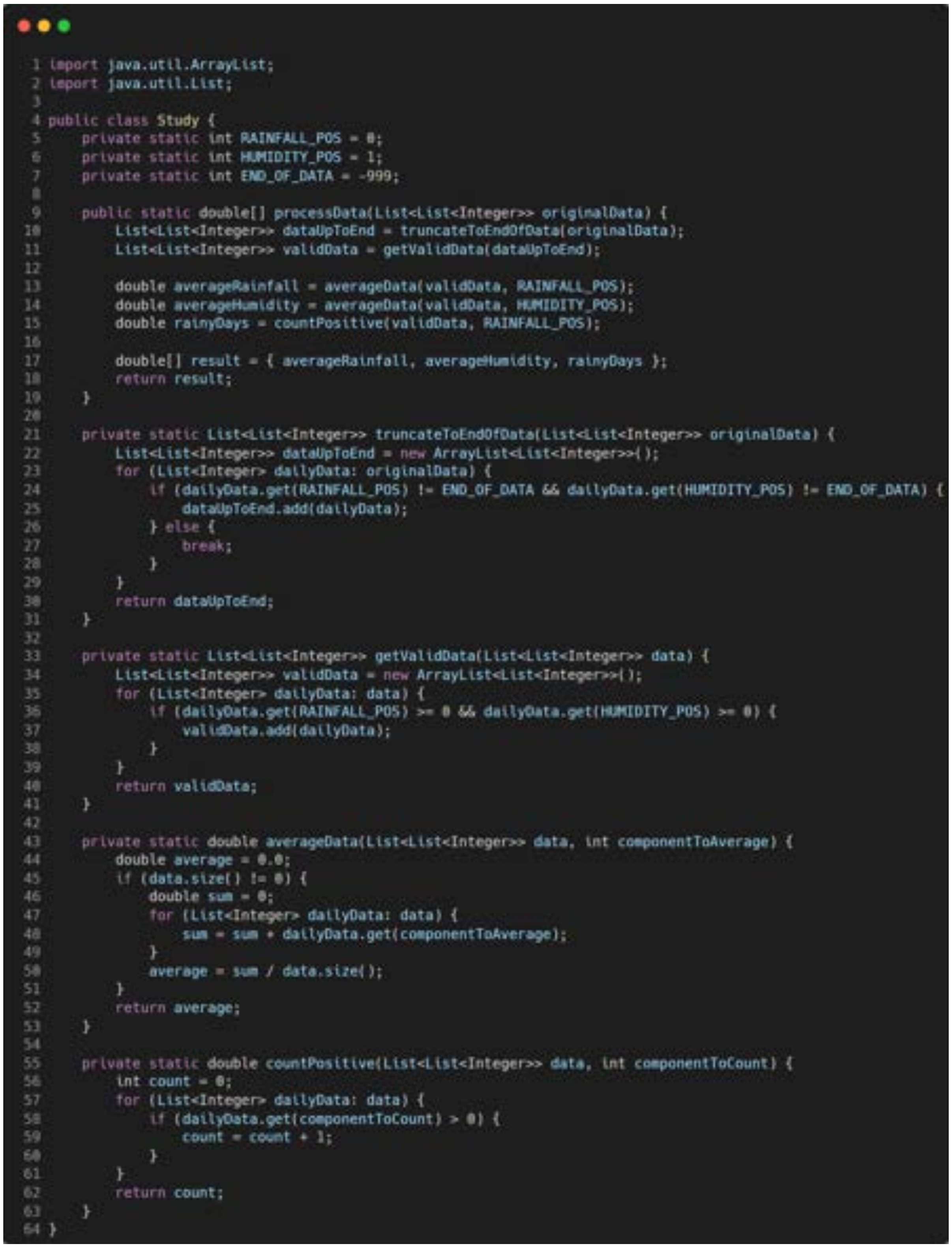
- ○ [-3, -3, -3]
- ○ [-3, -3, 2]
- [✓ ○ [0, 0, 0]]
- ○ [0, 0, 2]
- ○ [-3, -3, 0]



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Q3 Given the same code below. If the input `[[1,1], [1,1], [-999,-999], [1,1]]` is passed to the function `main()`, what is the output?

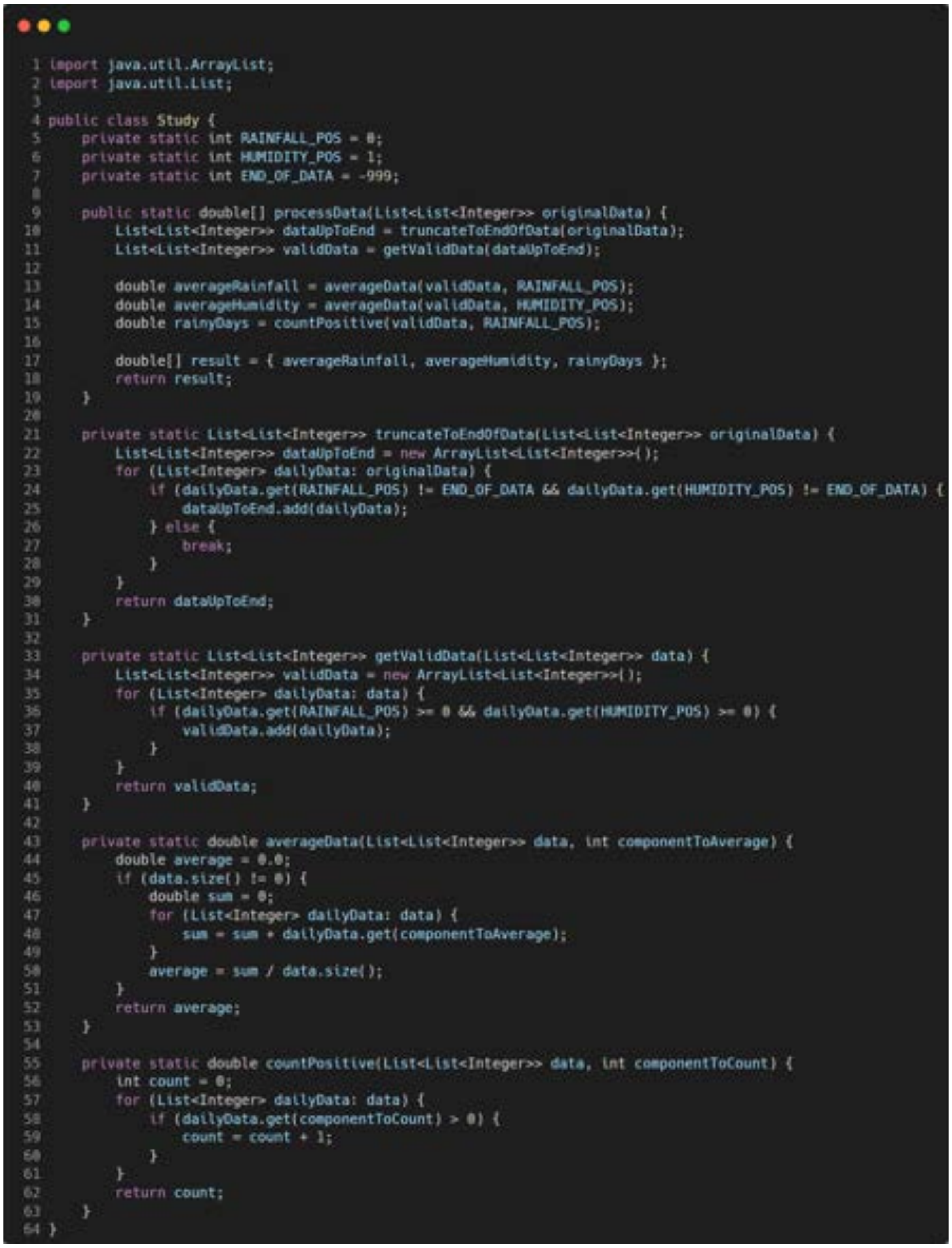
- ○ [1, 1, -999]
- ○ [2, 2, -999]
- ○ [3, 3, -999]
- ○ [0, 0, 0]
- [✓ ○ [1, 1, 2]]



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Q4 Given the same code below. What input to `main()` would produce the output `[0, 0, 0]`

- ○ [[0,0], [1,1], [1,2]]
- ○ [[0,0], [-1,-1], [1,2]]
- ○ [[-1,-1], [1,1], [1,2]]
- [✓ ○ [[0,-999], [0,0], [1,2]]]
- ○ [[0,0], [1,1], [1,-999]]

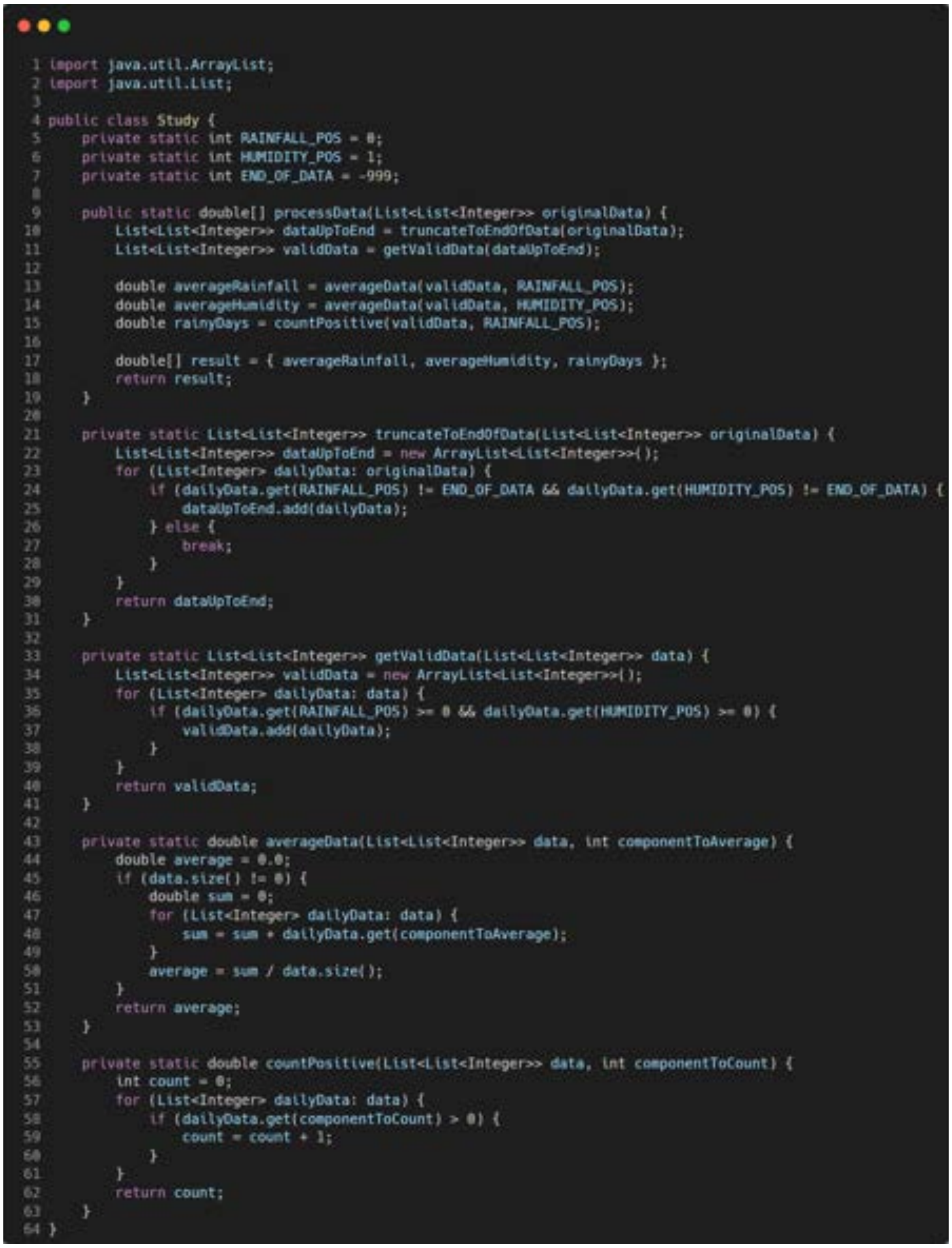


```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Q5

Given the same code below. Suppose you had to change to the code to ignore values larger than 100. Which of the following sets of lines would you need to modify (one or more lines) to achieve this change?

- ○ 9-19
- ○ 21-31
- [✓ ○ 33-41]
- ○ 43-53
- ○ 55-63

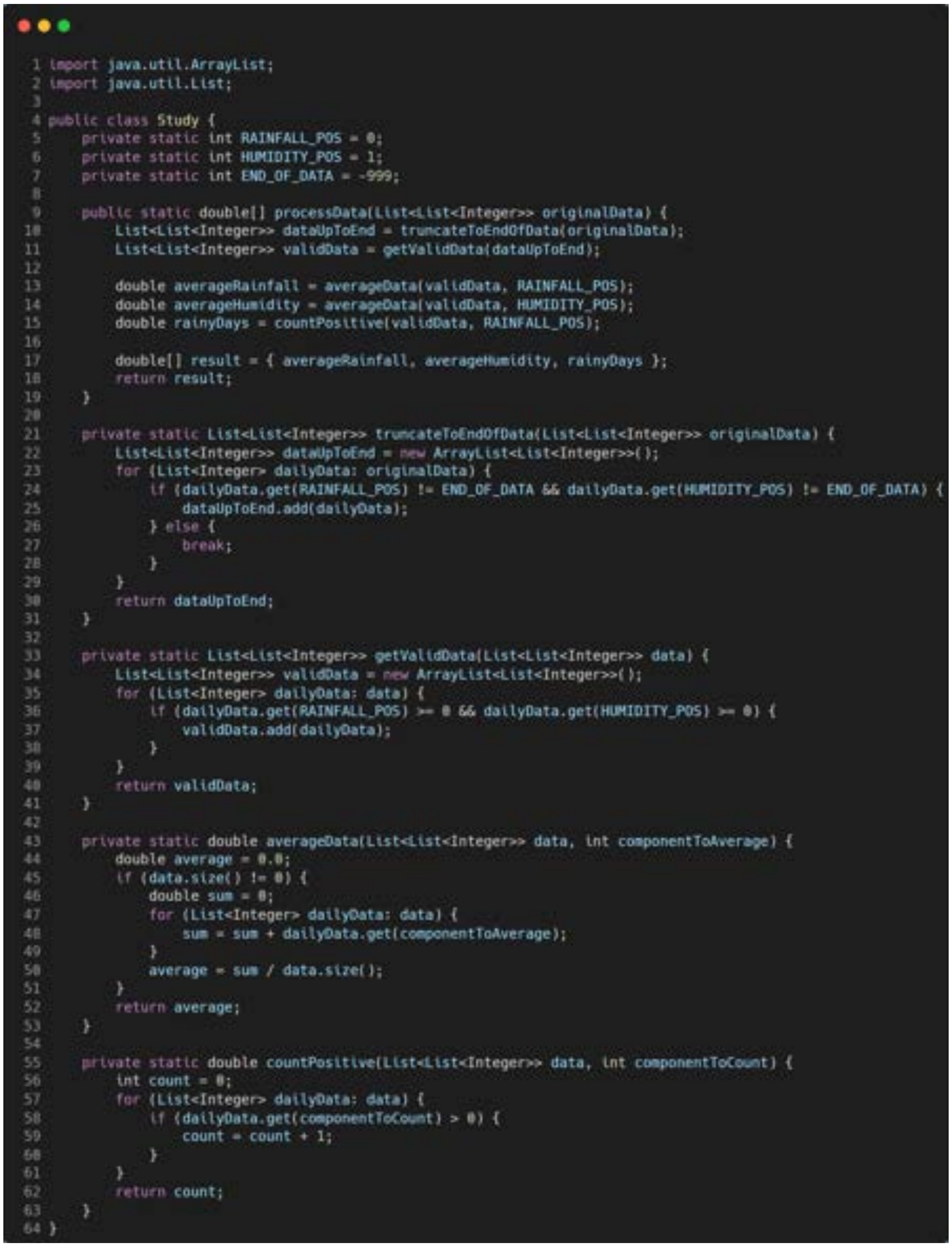


```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Given the same code below. How easy do you think this code is to understand?

- Very easy
- Somewhat easy
- Not easy, but not difficult
- Difficult
- Very difficult

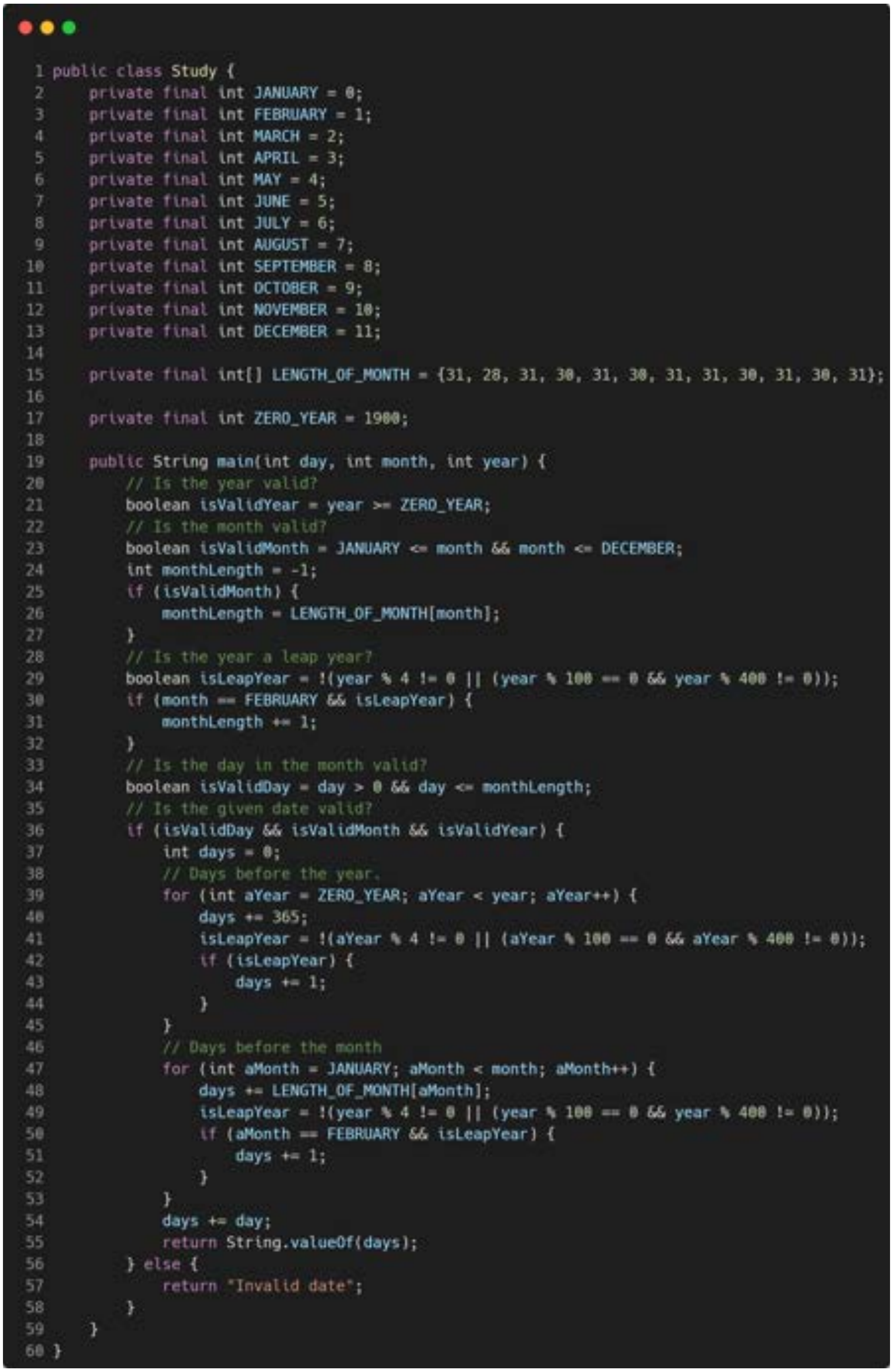
Comment on your response (optional)



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

Days SFV

Instructions: Study the code presented below. You will be asked a question about this code on the following page. Once you leave this page you will not be able to return, so spend as much time as you need to understand this code.
You may assume there are no errors in the code.



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

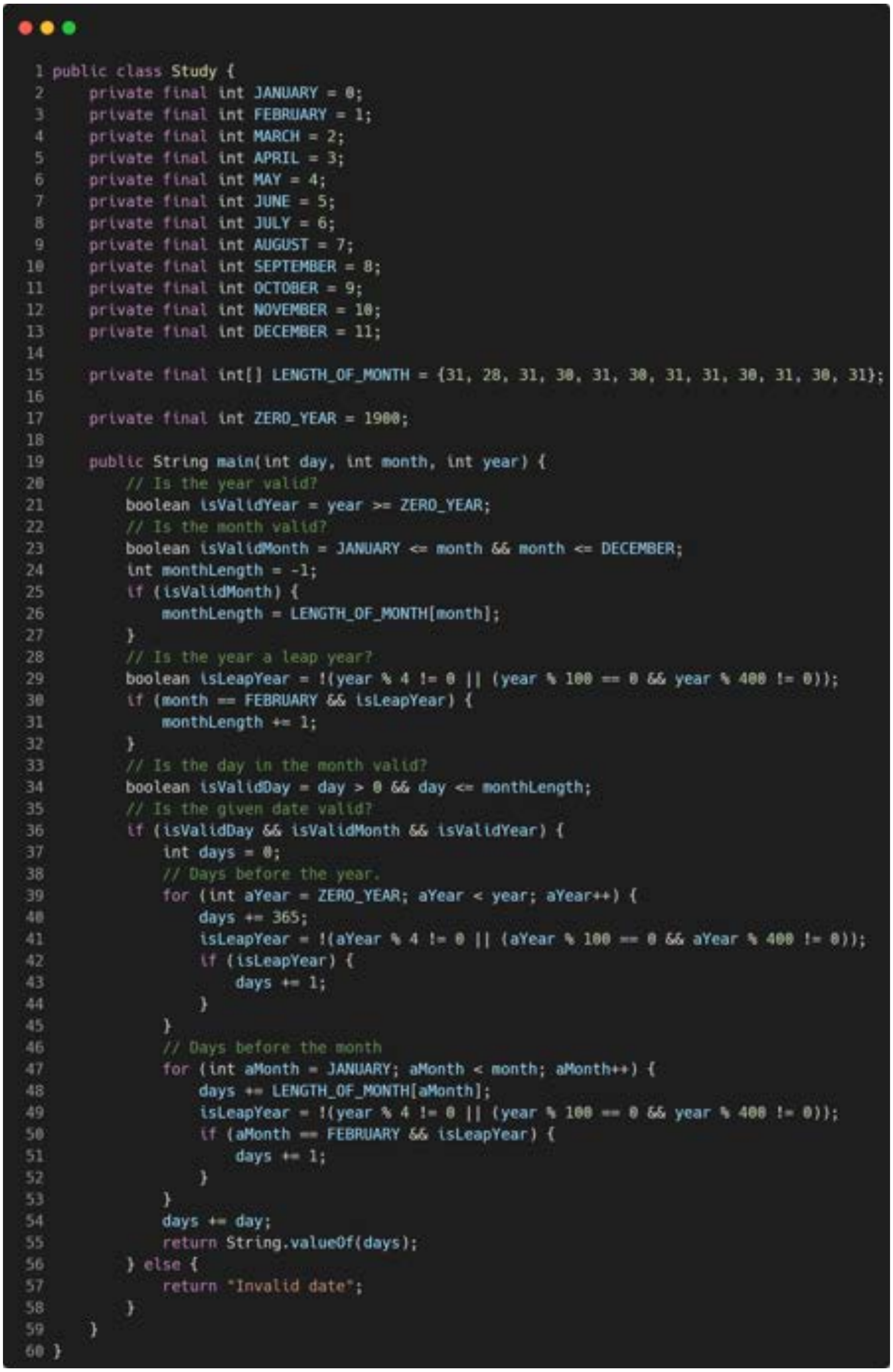
Write 2-3 sentences explaining what the code you have just studied on the previous page does. Your description should be at a high-level but still be complete.

Rate your confidence of the accuracy of your answer.

- ○ Low
- ○ Medium
- ○ High

Q1 Given the same code below. What is the value returned for main(1, JANUARY, 1900)? Note that 1900 is not a leap year.

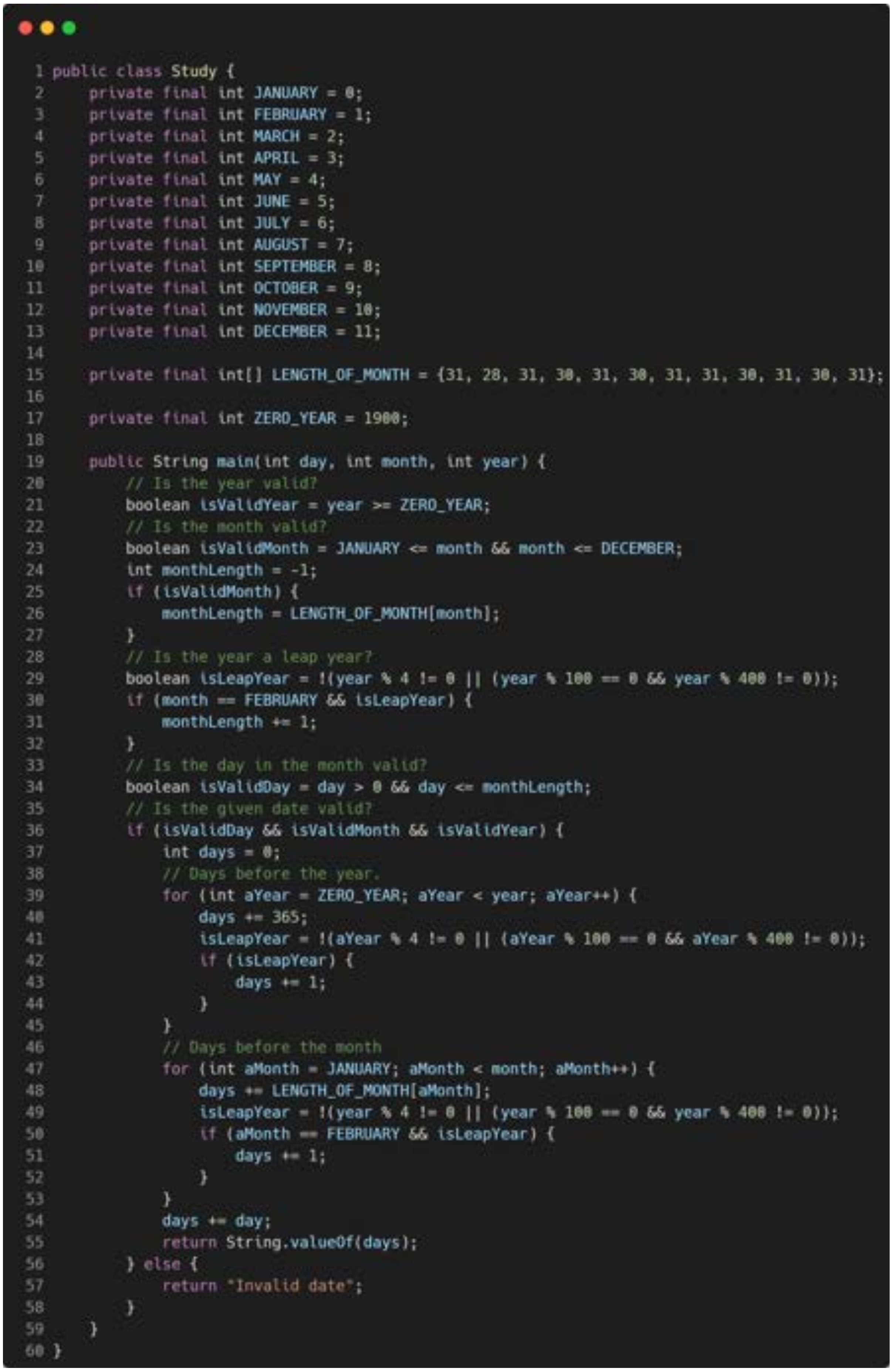
- [✓ ○ 1]
- ○ 365
- ○ 2
- ○ 0
- ○ Invalid date



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Q2 Given the same code below. What is the value returned for main(1, MARCH, 1900)? Note that 1900 is not a leap year.

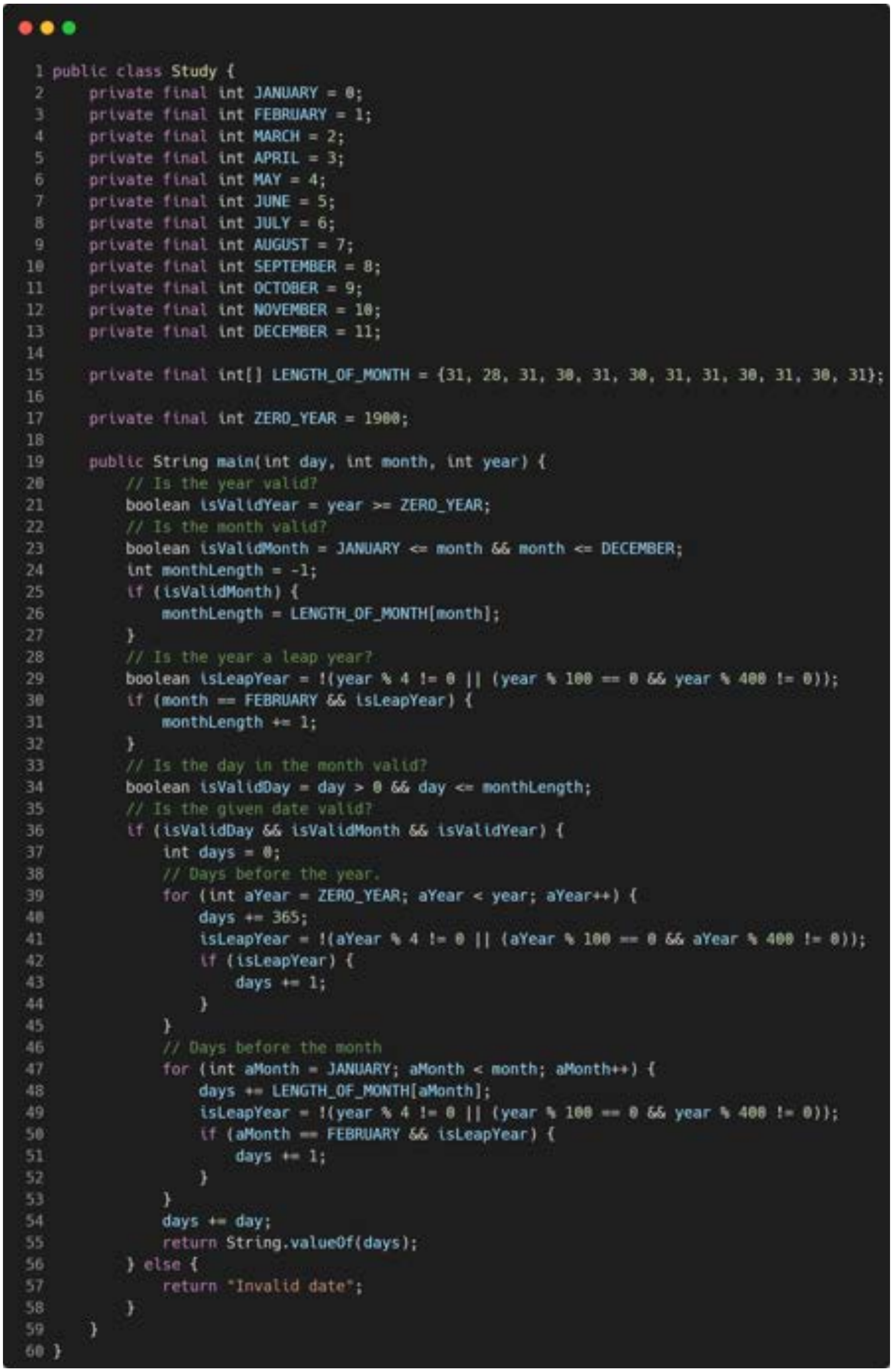
- [✓ 60]
- 61
- 62
- 425
- Invalid date



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Q3 Given the same code below. What is the value returned for main(31, APRIL, 1900)? Note that 1900 is not a leap year.

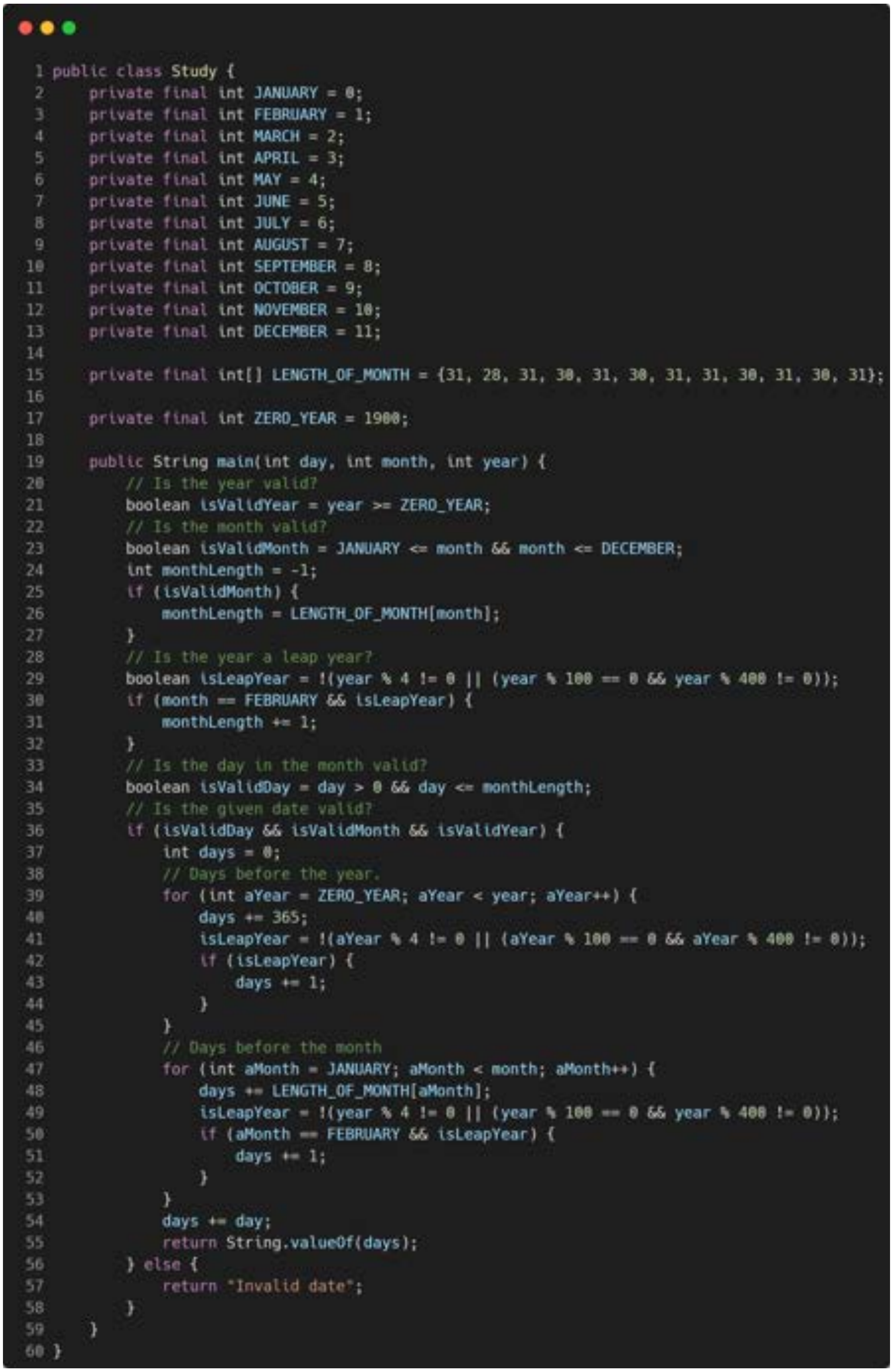
- 485
- 121
- 122
- 120
- [✓ Invalid date]



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Q4 Given the same code below. The value returned for main(1, JANUARY, 2024) is 45291, what is the value returned for main(1, MARCH, 2024)? Note that 2024 is a leap year.

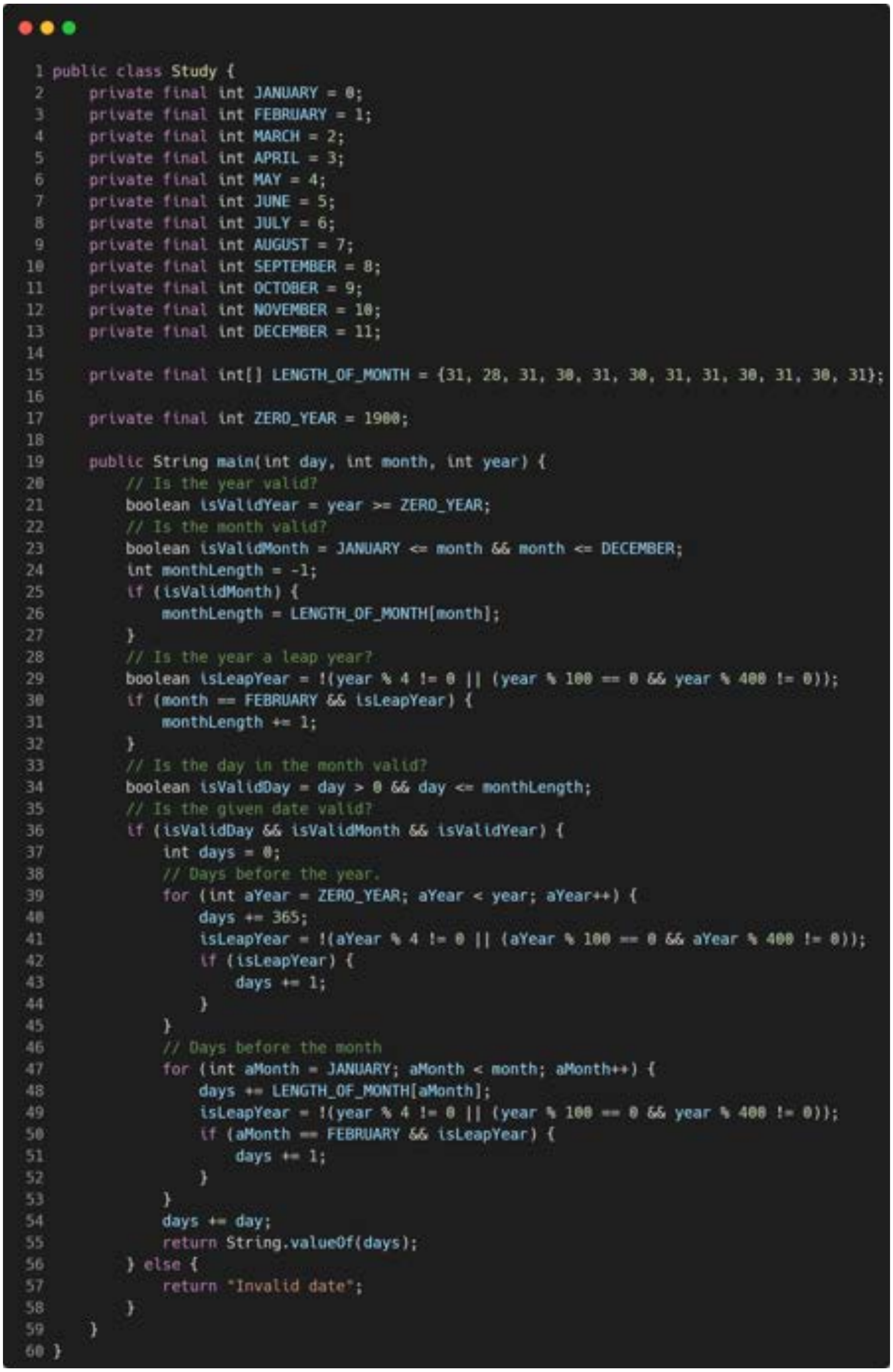
- [✓ 45351]
- 45352
- 45330
- 45350
- Invalid date



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Q5 Given the same code below. What input to main would produce the output 32? Note that 1900 is not a leap year.

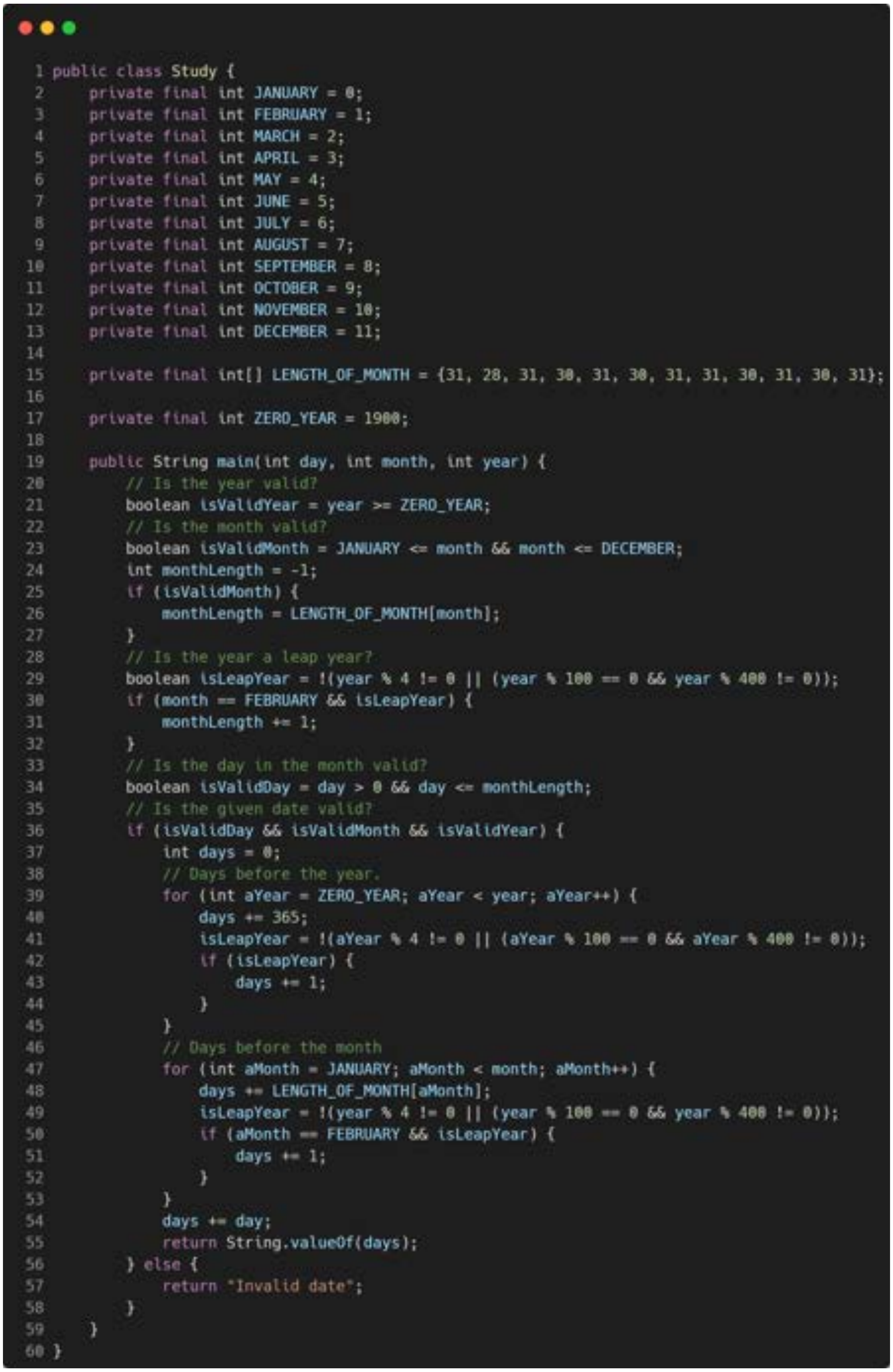
- [✓ 1, FEBRUARY, 1900]
- 3, FEBRUARY, 1900
- 2, FEBRUARY, 1900
- 31, JANUARY, 1900
- No input to main would produce 32



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Q6 Given the code below. If leap years had 31 days for JUNE instead of 29 days for FEBRUARY what lines would need to be changed?

- [✓ 30, 50]
- 29, 41, 49
- 29, 30, 41, 42, 49, 50
- 50
- 30

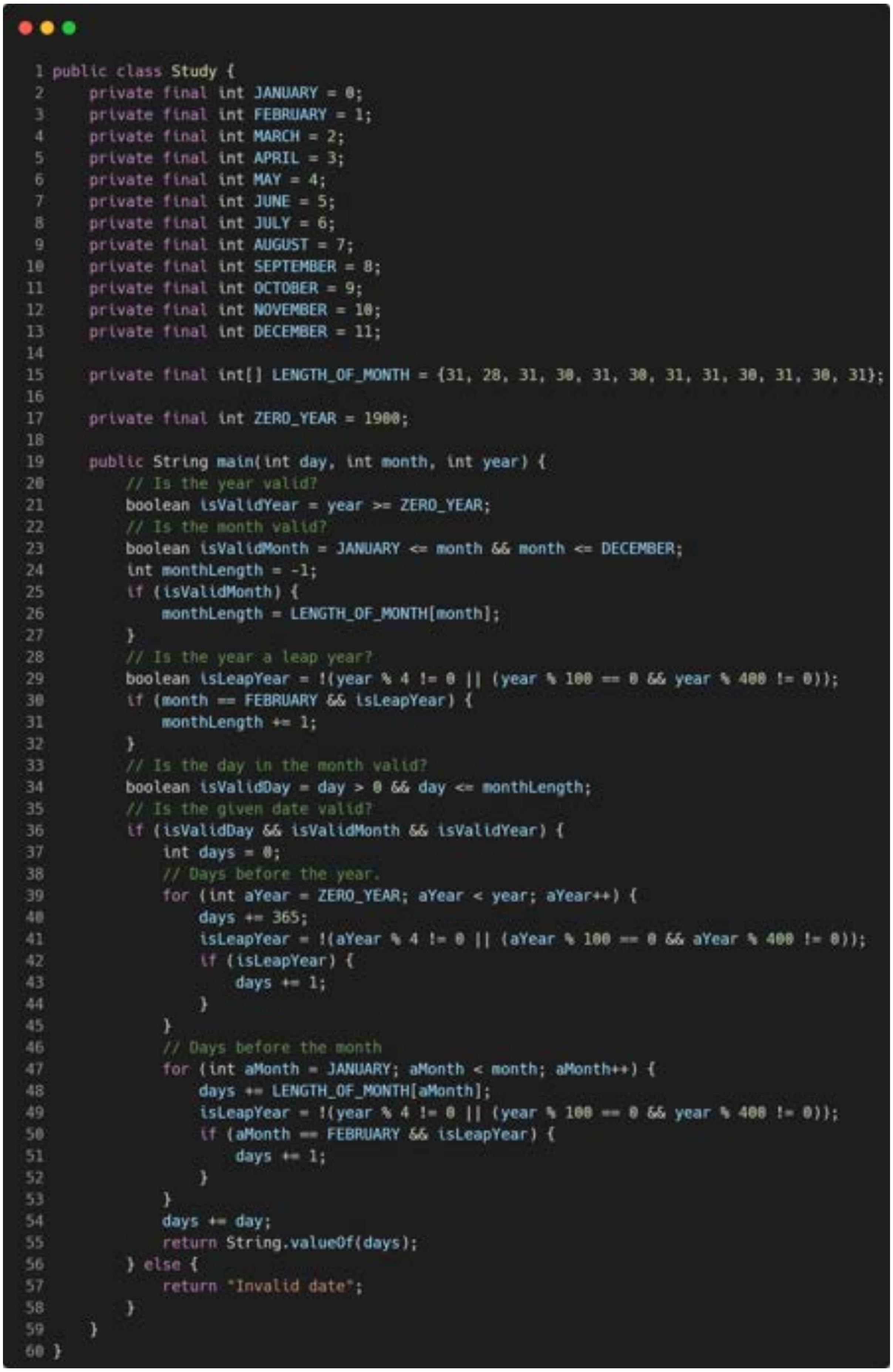


```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Given the same code below. How easy do you think this code is to understand?

- ○ Very easy
- ○ Somewhat easy
- ○ Not easy, but not difficult
- ○ Difficult
- ○ Very difficult

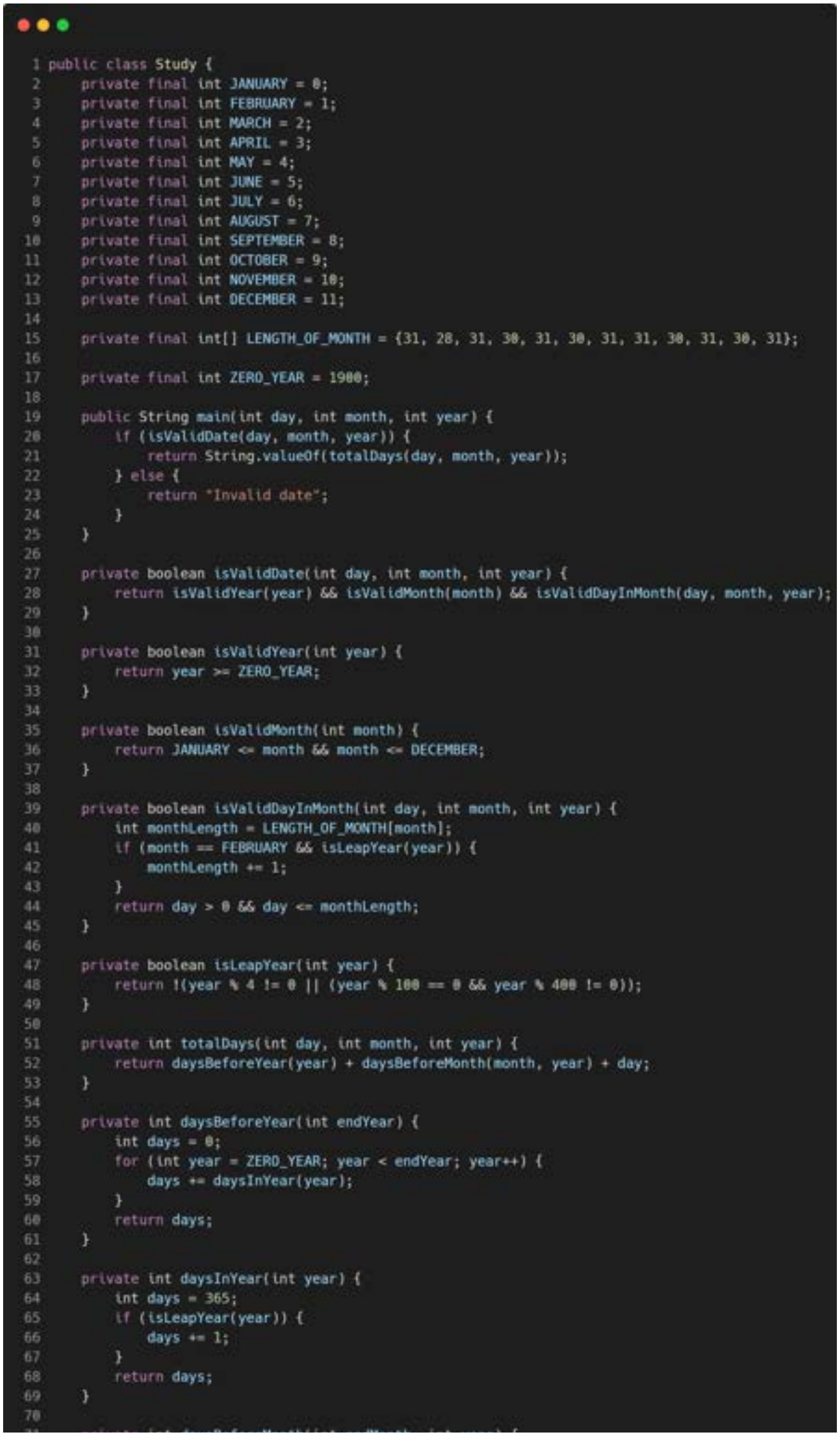
Comment on your response (optional)



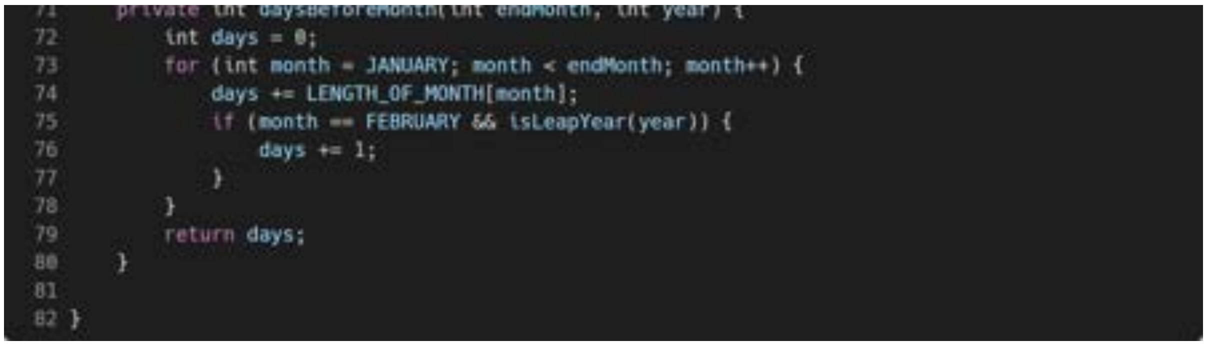
```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Days MFV

Instructions: Study the code presented below. You will be asked a question about this code on the following page. Once you leave this page you will not be able to return, so spend as much time as you need to understand this code.
You may assume there are no errors in the code.



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }

```



```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

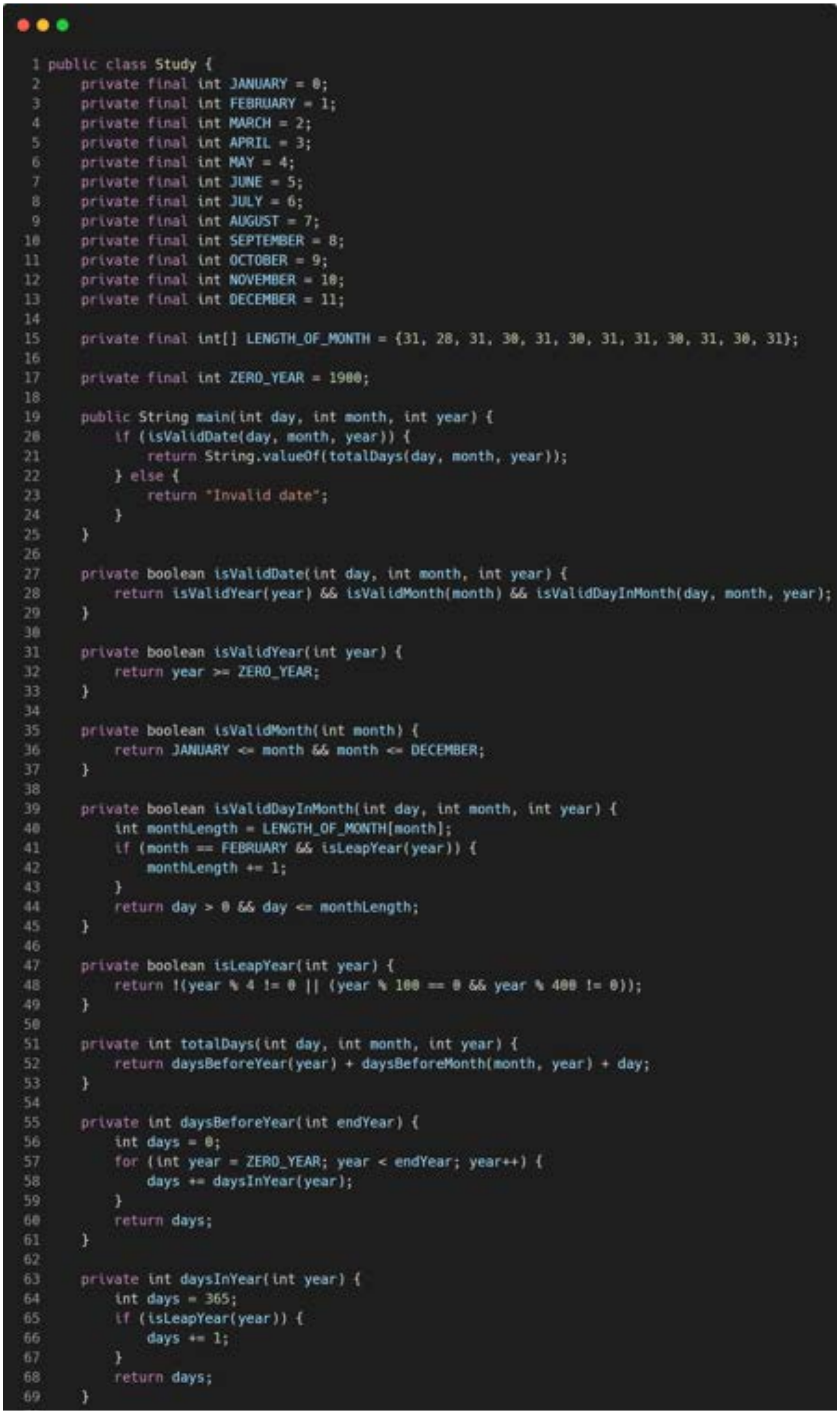
Write 2-3 sentences explaining what the code you have just studied on the previous page does. Your description should be at a high-level but still be complete.

Rate your confidence of the accuracy of your answer.

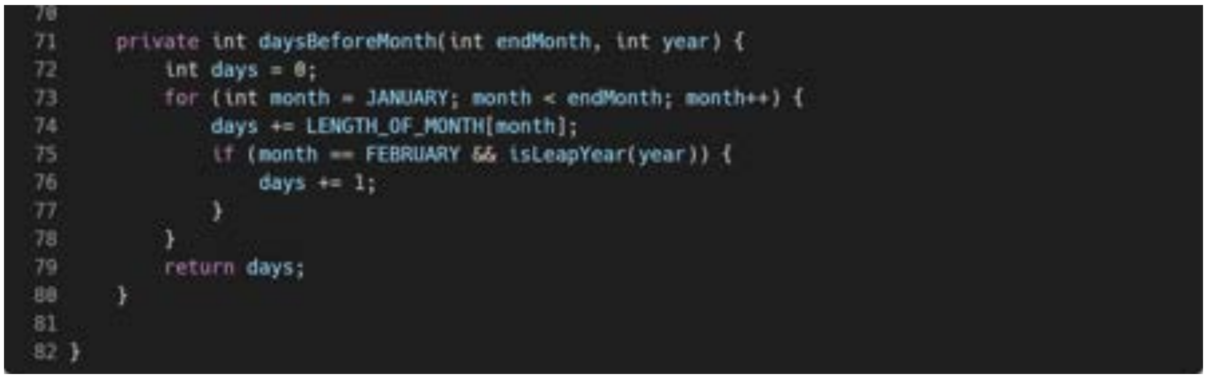
- Low
- Medium
- High

Q1 Given the same code below. What is the value returned for main(1, JANUARY, 1900)? Note that 1900 is not a leap year.

- [✓ 1]
- 365
- 2
- 0
- Invalid date



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```

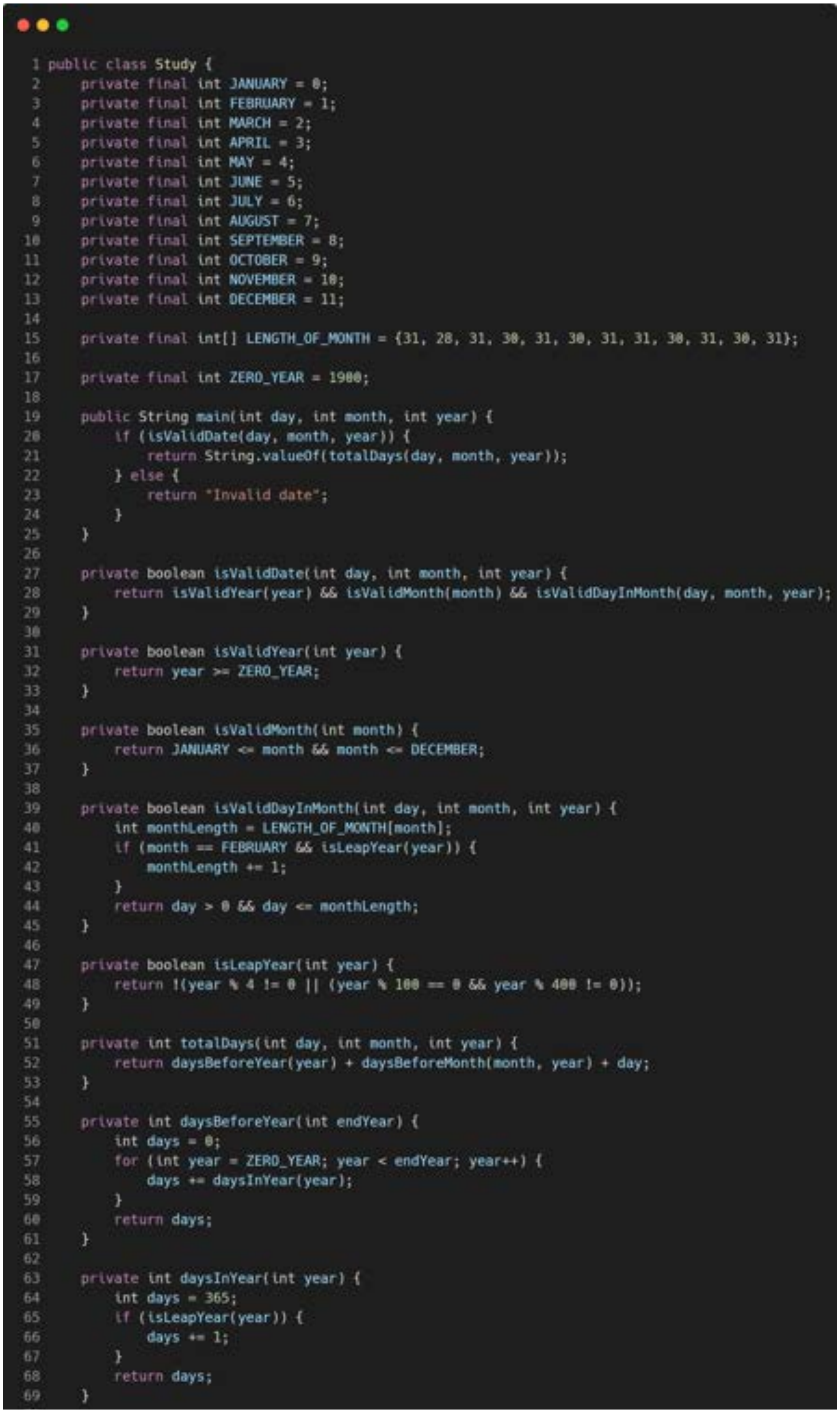


```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

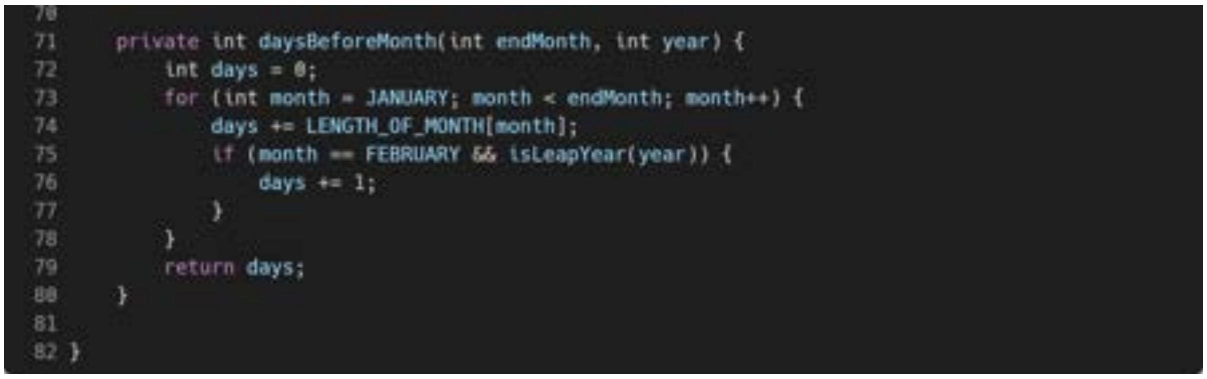
Q2

Given the same code below. What is the value returned for main(1, MARCH, 1900)? Note that 1900 is not a leap year.

- [✓ 60]
- 61
- 62
- 425
- Invalid date



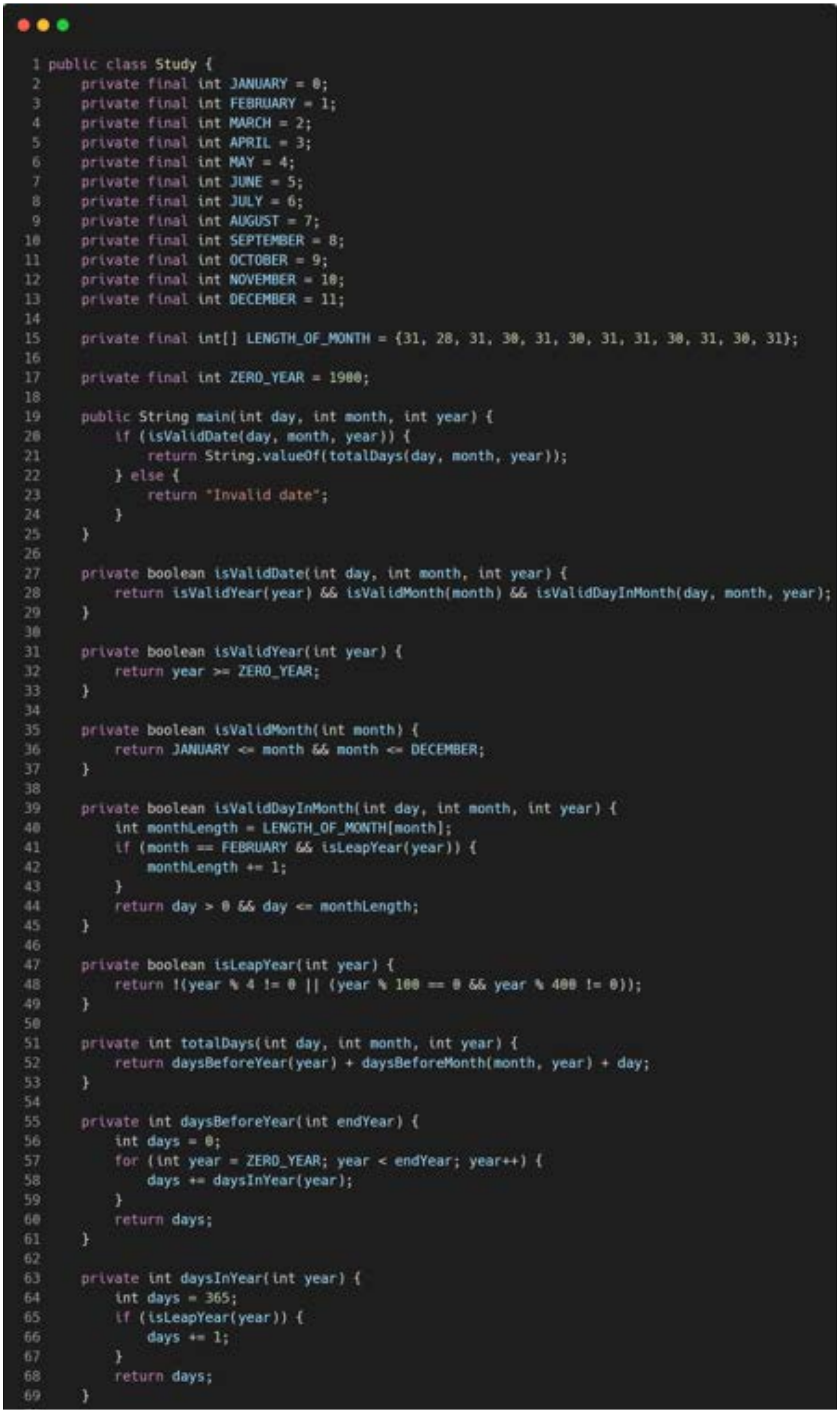
```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```



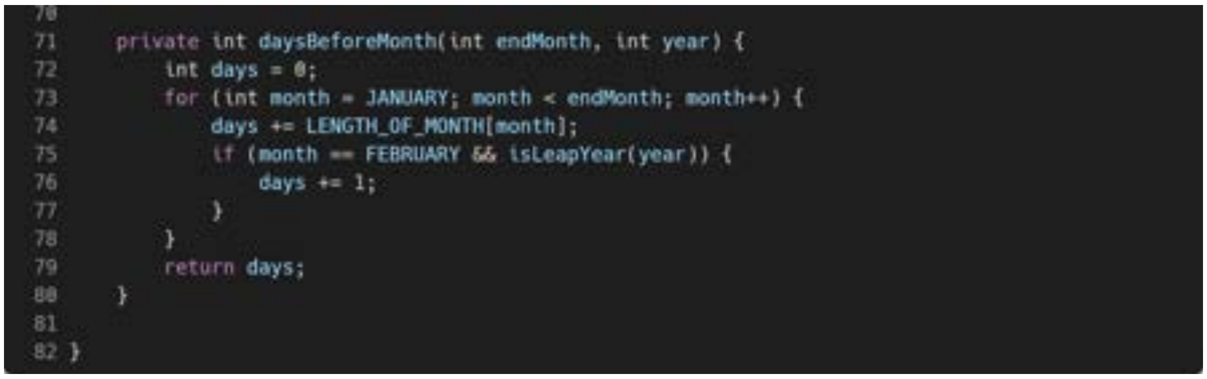
```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

Q3 Given the same code below. What is the value returned for main(31, APRIL, 1900)? Note that 1900 is not a leap year.

- 485
- 121
- 122
- 120
- [✓ Invalid date]



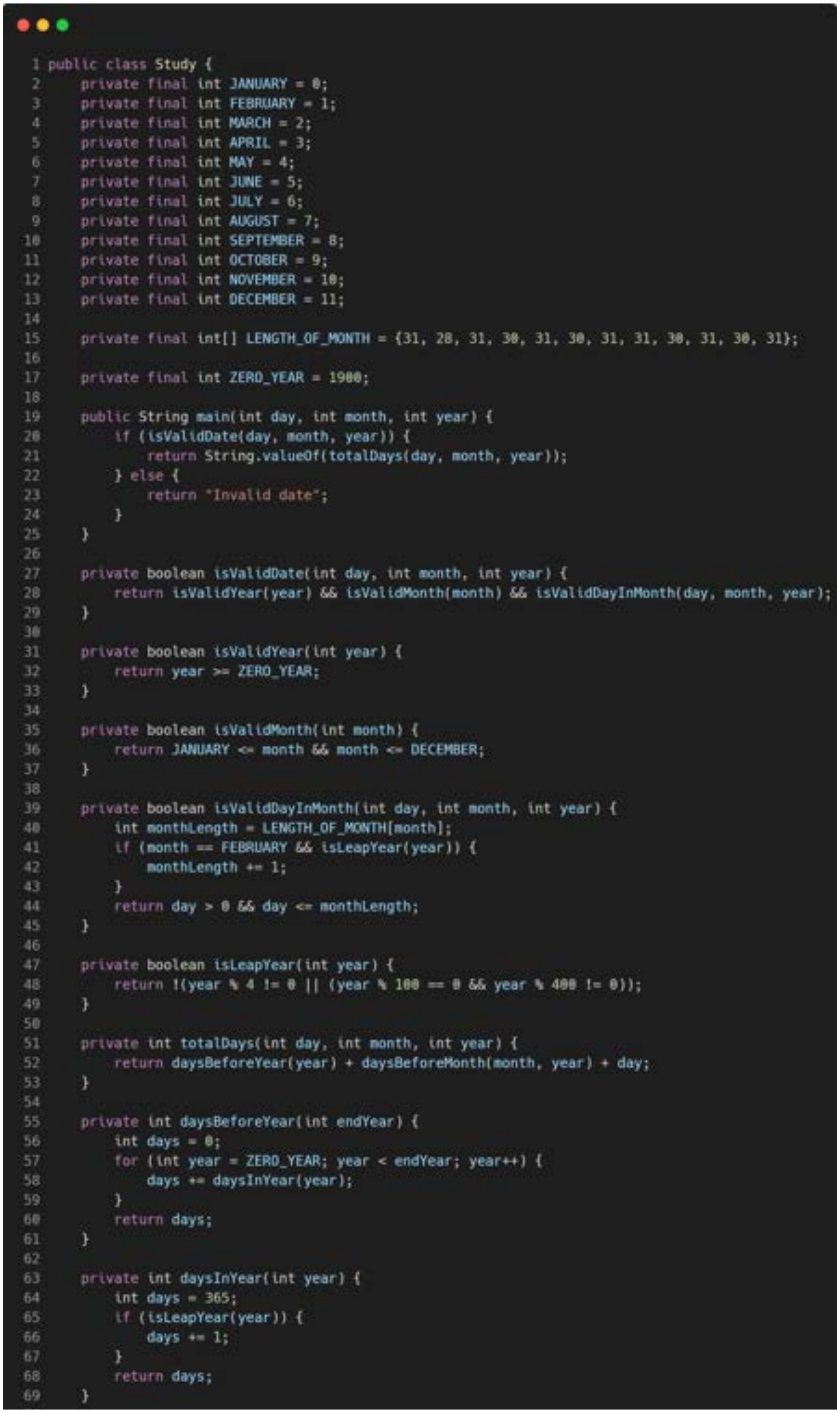
```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```



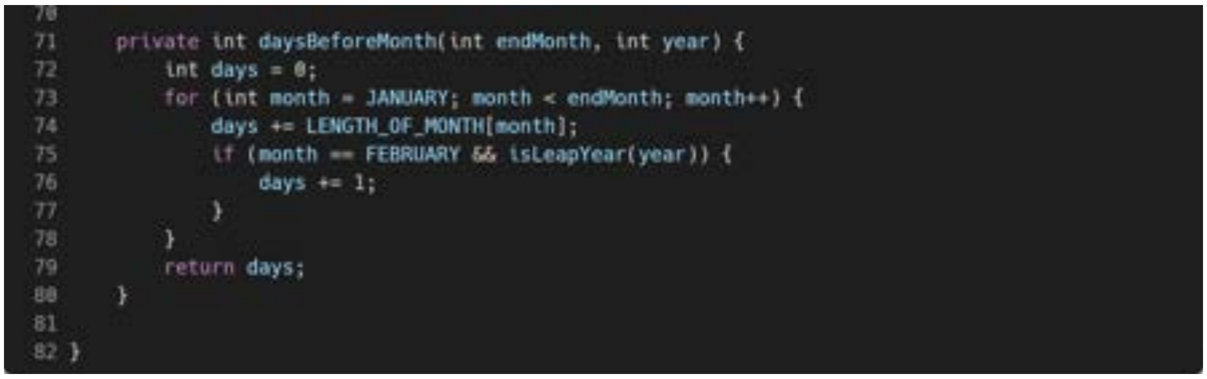
```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

Q4 Given the same code below. The value returned for main(1, JANUARY, 2024) is 45291, what is the value returned for main(1, MARCH, 2024)? Note that 2024 is a leap year.

- [✓ 45351]
- 45352
- 45330
- 45350
- Invalid date



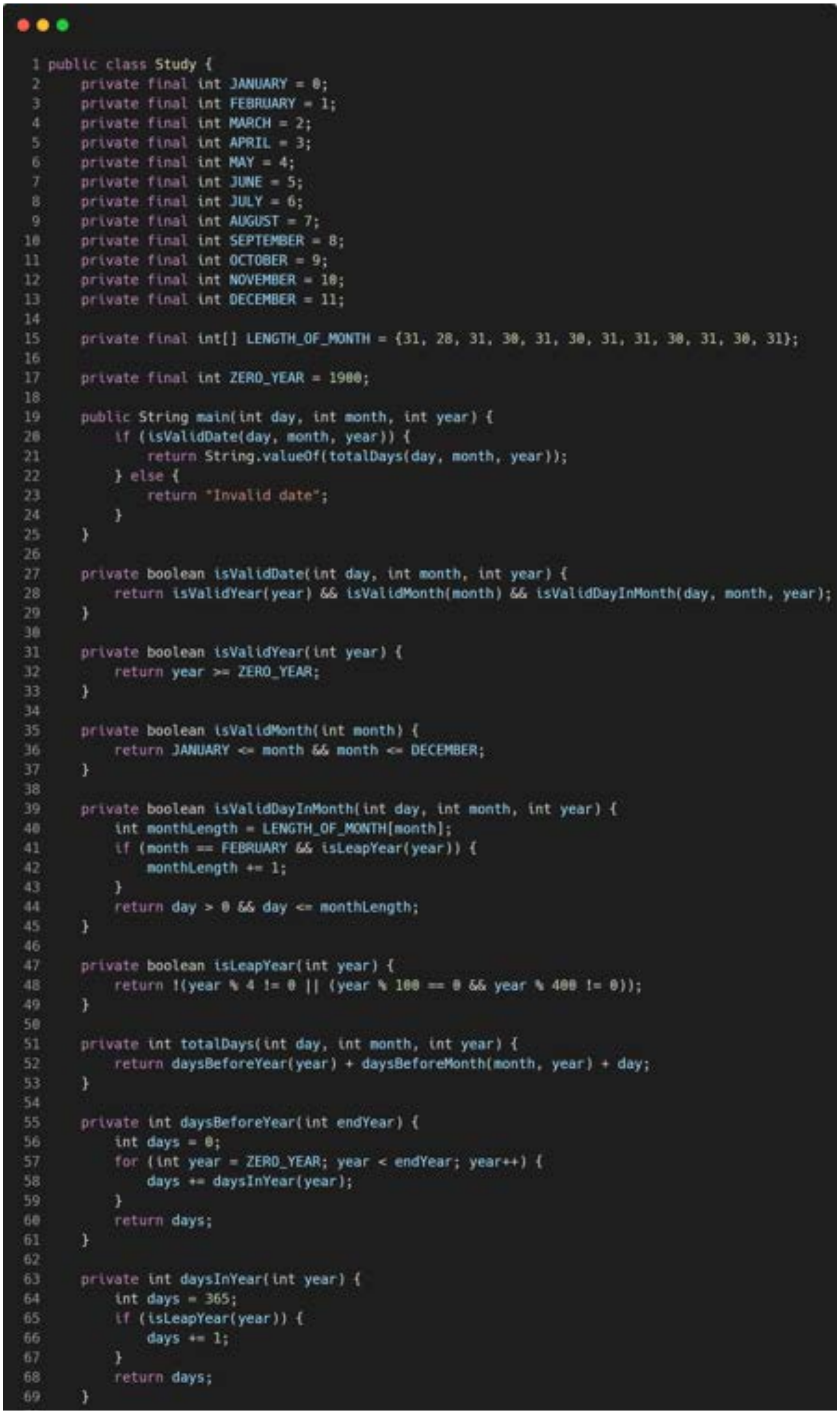
```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```



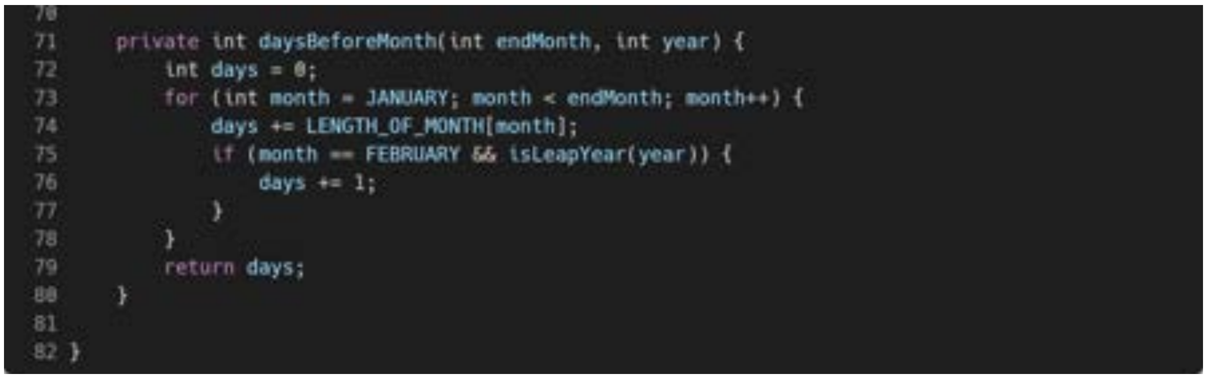
```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

Q5 Given the same code below. What input to main would produce the output 32? Note that 1900 is not a leap year.

- [✓ 1, FEBRUARY, 1900]
- 3, FEBRUARY, 1900
- 2, FEBRUARY, 1900
- 31, JANUARY, 1900
- No input to main would produce 32



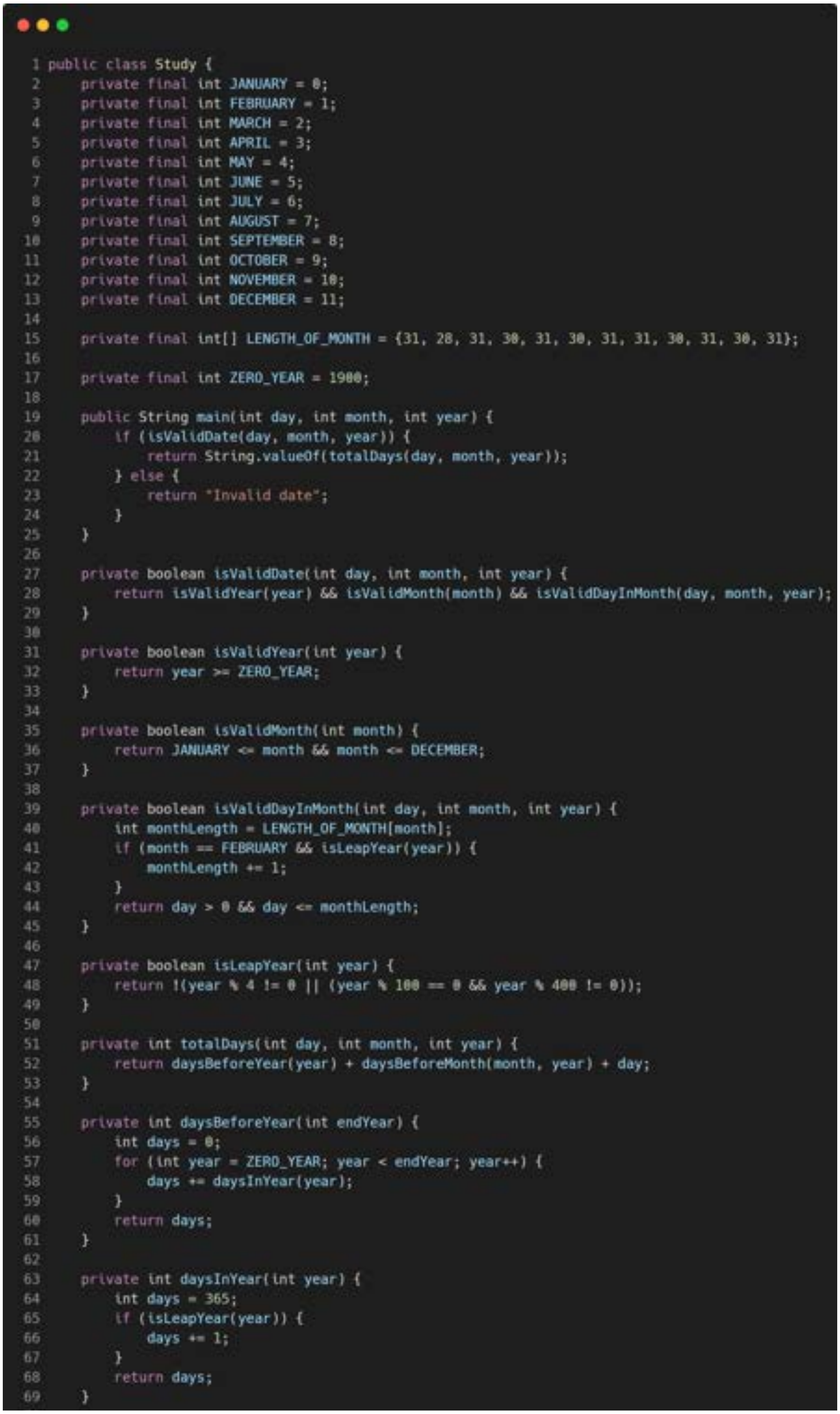
```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```



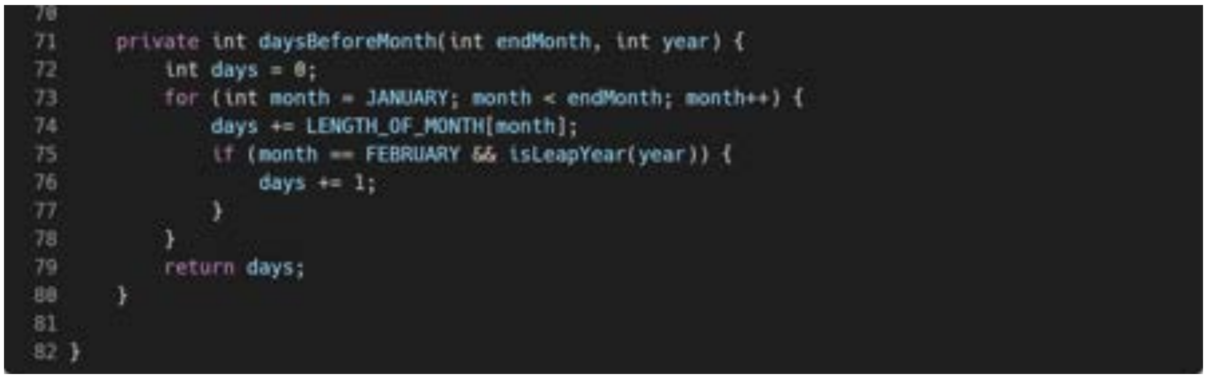
```
70
71      private int daysBeforeMonth(int endMonth, int year) {
72          int days = 0;
73          for (int month = JANUARY; month < endMonth; month++) {
74              days += LENGTH_OF_MONTH[month];
75              if (month == FEBRUARY && isLeapYear(year)) {
76                  days += 1;
77              }
78          }
79          return days;
80      }
81
82  }
```

Q6 Given the code below. If leap years had 31 days for JUNE instead of 29 days for FEBRUARY what lines would need to be changed?

- [✓ 41, 75]
- 41, 65, 75
- 41, 48, 65, 75
- 41
- 75



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```

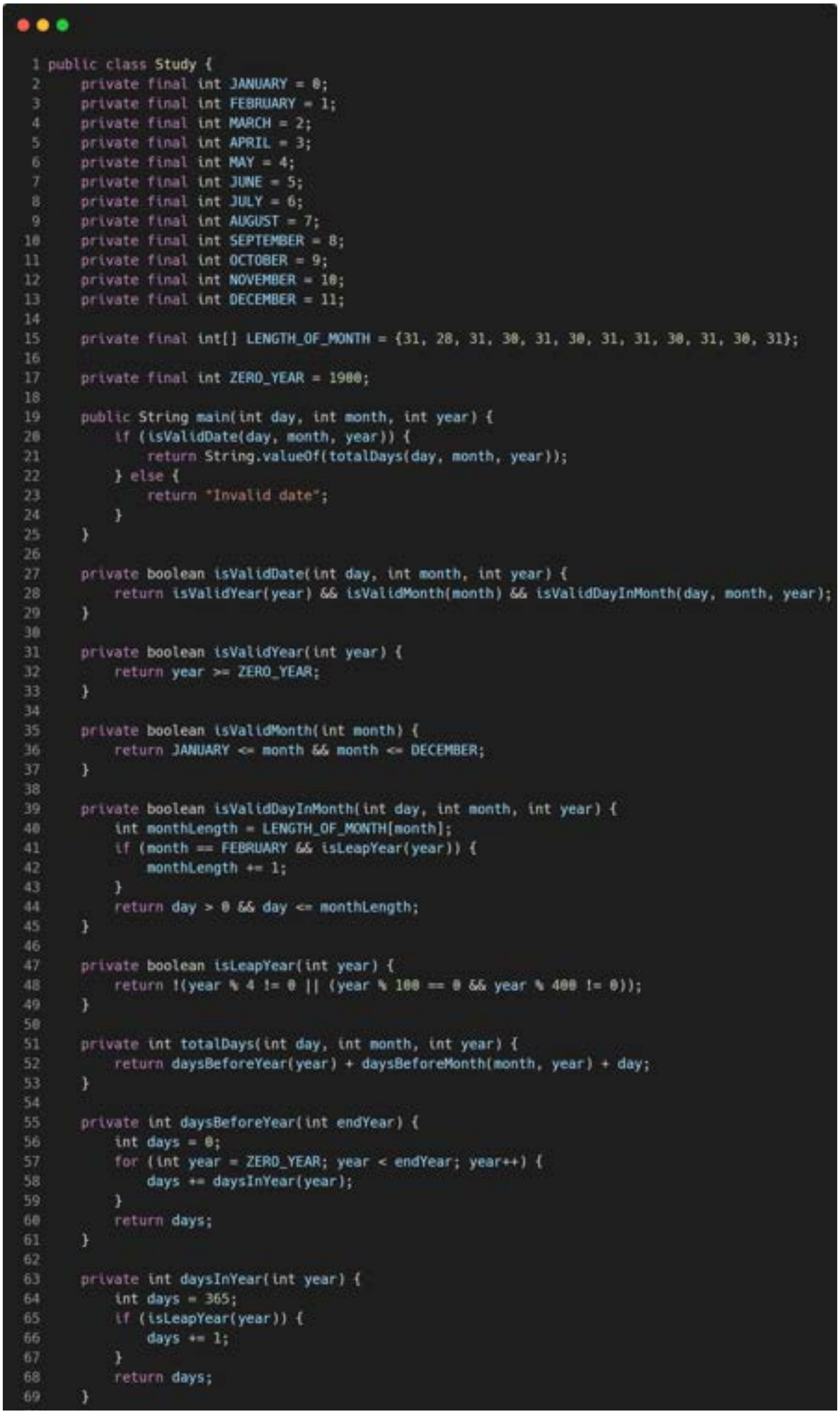


```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

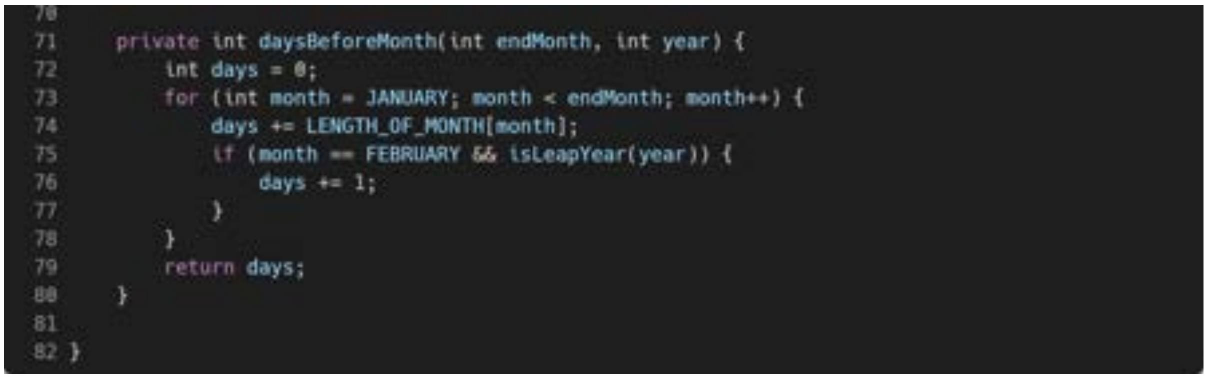
Given the same code below. How easy do you think this code is to understand?

- ○ Very easy
- ○ Somewhat easy
- ○ Not easy, but not difficult
- ○ Difficult
- ○ Very difficult

Comment on your response (optional)



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }
```



```java
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

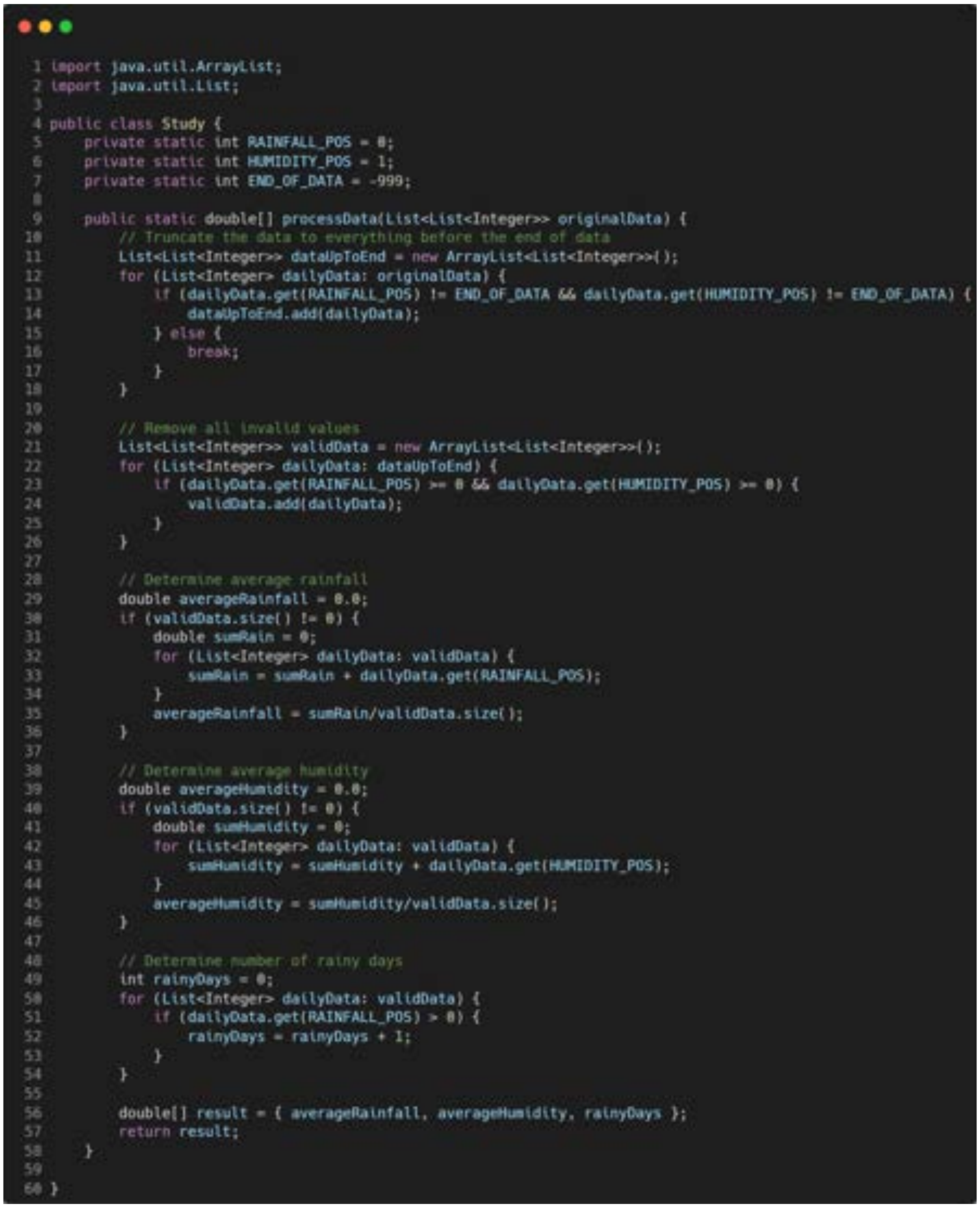
Weather Comparison

Given the same code and an alternative version below, which version do you think takes less effort to understand?

- ○ Version 1
- ○ Version 2
- ○ Both versions take the same effort

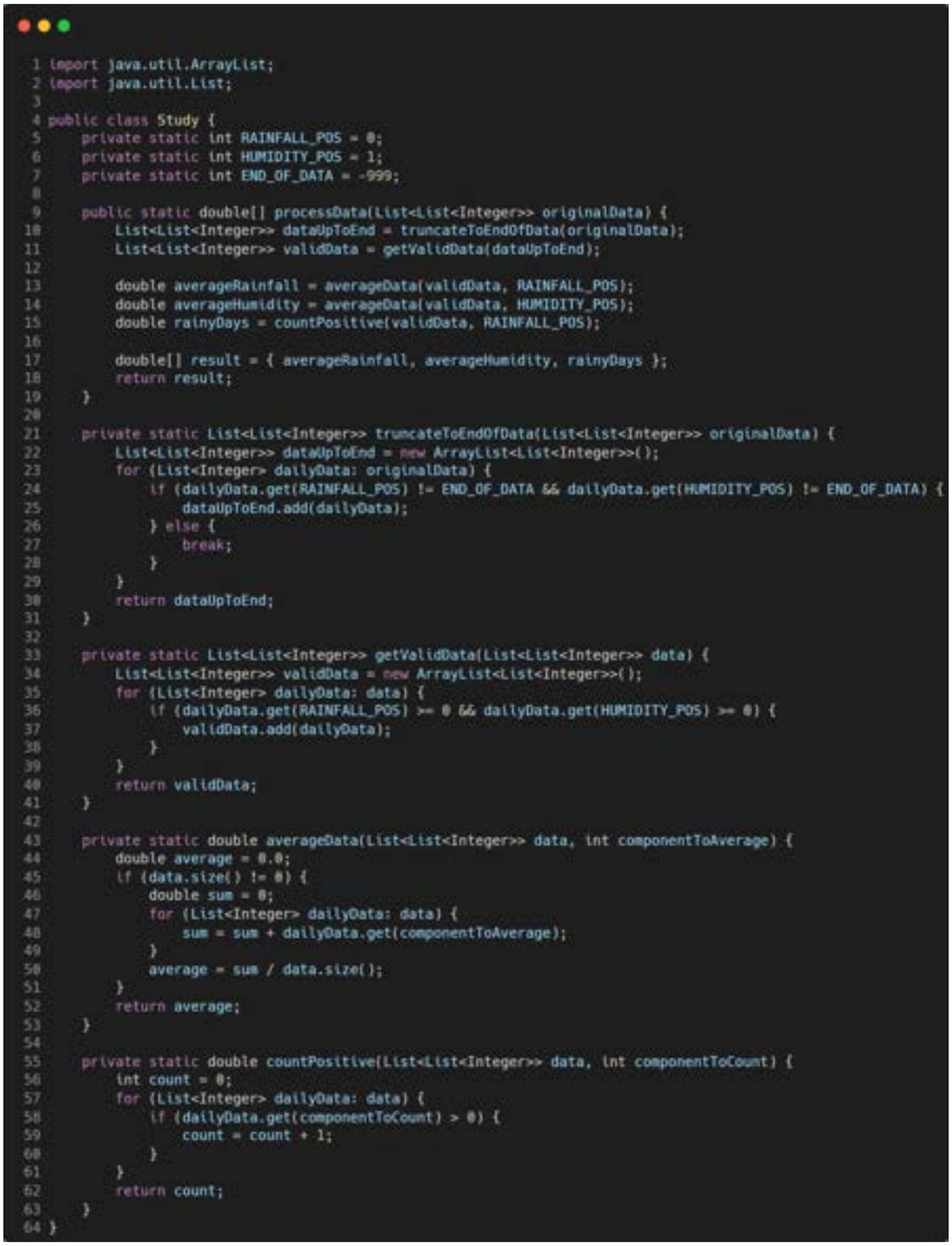
Please comment on why you think one version would take less effort, or the same.

Version 1



```
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        // Truncate the data to everything before the end of data
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }

        // Remove all invalid values
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: dataUpToEnd) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }

        // Determine average rainfall
        double averageRainfall = 0.0;
        if (validData.size() != 0) {
            double sumRain = 0;
            for (List<Integer> dailyData: validData) {
                sumRain = sumRain + dailyData.get(RAINFALL_POS);
            }
            averageRainfall = sumRain/validData.size();
        }

        // Determine average humidity
        double averageHumidity = 0.0;
        if (validData.size() != 0) {
            double sumHumidity = 0;
            for (List<Integer> dailyData: validData) {
                sumHumidity = sumHumidity + dailyData.get(HUMIDITY_POS);
            }
            averageHumidity = sumHumidity/validData.size();
        }

        // Determine number of rainy days
        int rainyDays = 0;
        for (List<Integer> dailyData: validData) {
            if (dailyData.get(RAINFALL_POS) > 0) {
                rainyDays = rainyDays + 1;
            }
        }

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

}
```

Version 2



```java
import java.util.ArrayList;
import java.util.List;

public class Study {
    private static int RAINFALL_POS = 0;
    private static int HUMIDITY_POS = 1;
    private static int END_OF_DATA = -999;

    public static double[] processData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = truncateToEndOfData(originalData);
        List<List<Integer>> validData = getValidData(dataUpToEnd);

        double averageRainfall = averageData(validData, RAINFALL_POS);
        double averageHumidity = averageData(validData, HUMIDITY_POS);
        double rainyDays = countPositive(validData, RAINFALL_POS);

        double[] result = { averageRainfall, averageHumidity, rainyDays };
        return result;
    }

    private static List<List<Integer>> truncateToEndOfData(List<List<Integer>> originalData) {
        List<List<Integer>> dataUpToEnd = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: originalData) {
            if (dailyData.get(RAINFALL_POS) != END_OF_DATA && dailyData.get(HUMIDITY_POS) != END_OF_DATA) {
                dataUpToEnd.add(dailyData);
            } else {
                break;
            }
        }
        return dataUpToEnd;
    }

    private static List<List<Integer>> getValidData(List<List<Integer>> data) {
        List<List<Integer>> validData = new ArrayList<List<Integer>>();
        for (List<Integer> dailyData: data) {
            if (dailyData.get(RAINFALL_POS) >= 0 && dailyData.get(HUMIDITY_POS) >= 0) {
                validData.add(dailyData);
            }
        }
        return validData;
    }

    private static double averageData(List<List<Integer>> data, int componentToAverage) {
        double average = 0.0;
        if (data.size() != 0) {
            double sum = 0;
            for (List<Integer> dailyData: data) {
                sum = sum + dailyData.get(componentToAverage);
            }
            average = sum / data.size();
        }
        return average;
    }

    private static double countPositive(List<List<Integer>> data, int componentToCount) {
        int count = 0;
        for (List<Integer> dailyData: data) {
            if (dailyData.get(componentToCount) > 0) {
                count = count + 1;
            }
        }
        return count;
    }
}
```

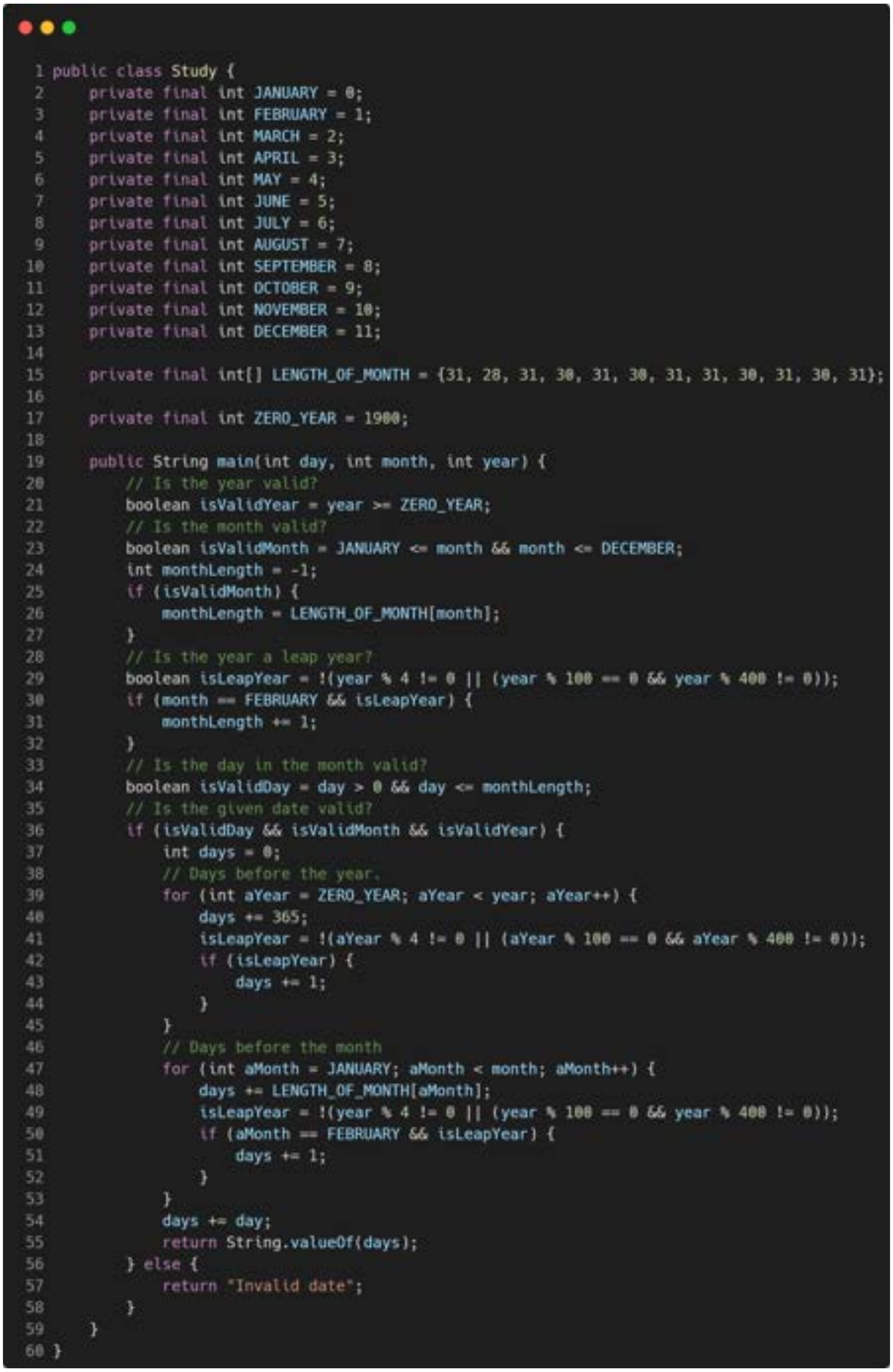
Days Comparison

Given the same code and an alternative version below, which version do you think takes less effort to understand?

- ○ Version 1
- ○ Version 2
- ○ Both versions take the same effort

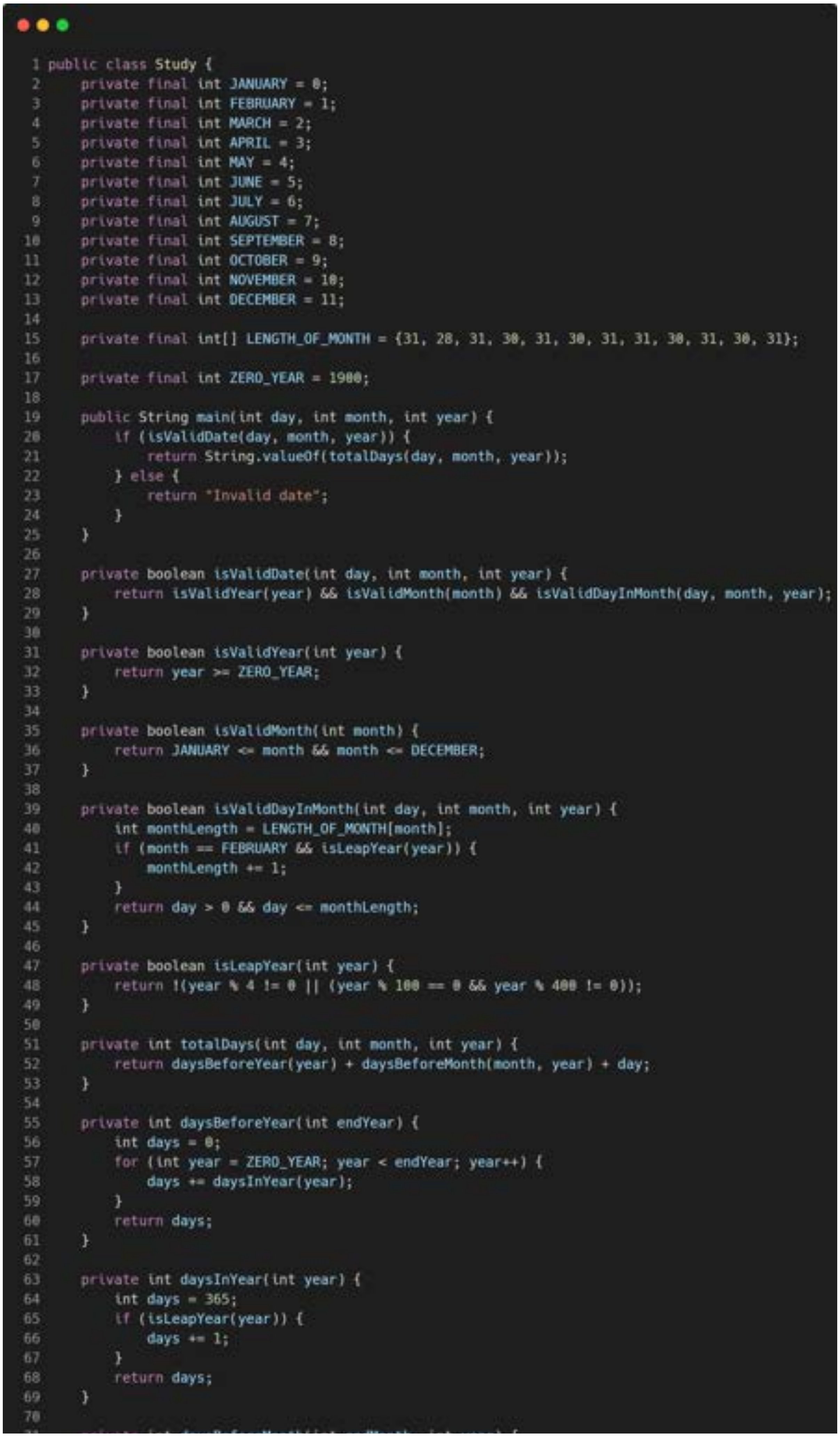
Please comment on why you think one version would take less effort, or the same.

Version 1

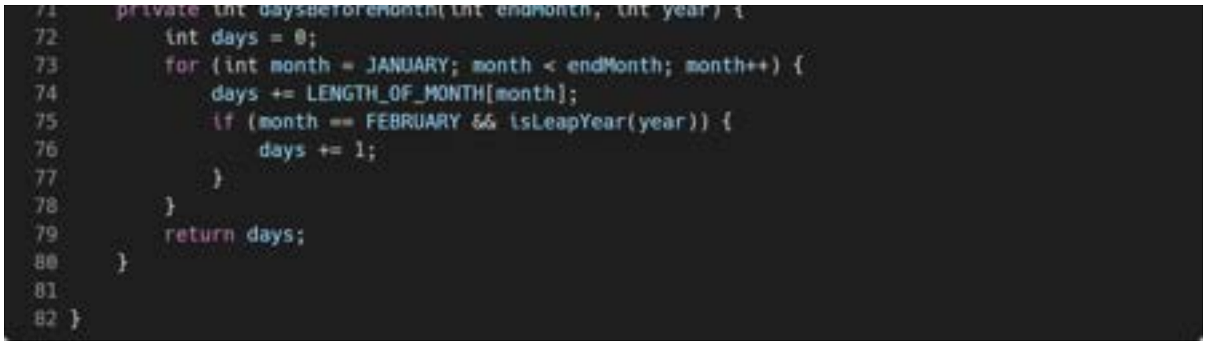


```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        // Is the year valid?
        boolean isValidYear = year >= ZERO_YEAR;
        // Is the month valid?
        boolean isValidMonth = JANUARY <= month && month <= DECEMBER;
        int monthLength = -1;
        if (isValidMonth) {
            monthLength = LENGTH_OF_MONTH[month];
        }
        // Is the year a leap year?
        boolean isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
        if (month == FEBRUARY && isLeapYear) {
            monthLength += 1;
        }
        // Is the day in the month valid?
        boolean isValidDay = day > 0 && day <= monthLength;
        // Is the given date valid?
        if (isValidDay && isValidMonth && isValidYear) {
            int days = 0;
            // Days before the year.
            for (int aYear = ZERO_YEAR; aYear < year; aYear++) {
                days += 365;
                isLeapYear = !(aYear % 4 != 0 || (aYear % 100 == 0 && aYear % 400 != 0));
                if (isLeapYear) {
                    days += 1;
                }
            }
            // Days before the month
            for (int aMonth = JANUARY; aMonth < month; aMonth++) {
                days += LENGTH_OF_MONTH[aMonth];
                isLeapYear = !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
                if (aMonth == FEBRUARY && isLeapYear) {
                    days += 1;
                }
            }
            days += day;
            return String.valueOf(days);
        } else {
            return "Invalid date";
        }
    }
}
```

Version 2



```java
public class Study {
    private final int JANUARY = 0;
    private final int FEBRUARY = 1;
    private final int MARCH = 2;
    private final int APRIL = 3;
    private final int MAY = 4;
    private final int JUNE = 5;
    private final int JULY = 6;
    private final int AUGUST = 7;
    private final int SEPTEMBER = 8;
    private final int OCTOBER = 9;
    private final int NOVEMBER = 10;
    private final int DECEMBER = 11;

    private final int[] LENGTH_OF_MONTH = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

    private final int ZERO_YEAR = 1900;

    public String main(int day, int month, int year) {
        if (isValidDate(day, month, year)) {
            return String.valueOf(totalDays(day, month, year));
        } else {
            return "Invalid date";
        }
    }

    private boolean isValidDate(int day, int month, int year) {
        return isValidYear(year) && isValidMonth(month) && isValidDayInMonth(day, month, year);
    }

    private boolean isValidYear(int year) {
        return year >= ZERO_YEAR;
    }

    private boolean isValidMonth(int month) {
        return JANUARY <= month && month <= DECEMBER;
    }

    private boolean isValidDayInMonth(int day, int month, int year) {
        int monthLength = LENGTH_OF_MONTH[month];
        if (month == FEBRUARY && isLeapYear(year)) {
            monthLength += 1;
        }
        return day > 0 && day <= monthLength;
    }

    private boolean isLeapYear(int year) {
        return !(year % 4 != 0 || (year % 100 == 0 && year % 400 != 0));
    }

    private int totalDays(int day, int month, int year) {
        return daysBeforeYear(year) + daysBeforeMonth(month, year) + day;
    }

    private int daysBeforeYear(int endYear) {
        int days = 0;
        for (int year = ZERO_YEAR; year < endYear; year++) {
            days += daysInYear(year);
        }
        return days;
    }

    private int daysInYear(int year) {
        int days = 365;
        if (isLeapYear(year)) {
            days += 1;
        }
        return days;
    }

```



```
    private int daysBeforeMonth(int endMonth, int year) {
        int days = 0;
        for (int month = JANUARY; month < endMonth; month++) {
            days += LENGTH_OF_MONTH[month];
            if (month == FEBRUARY && isLeapYear(year)) {
                days += 1;
            }
        }
        return days;
    }

}
```

Thank you

Thank you for taking part in this research. We appreciate your time and value your response in the questionnaire.

Please provide any reflections or feedback you have about this study.

Powered by Qualtrics