Part 3

Discovering errors

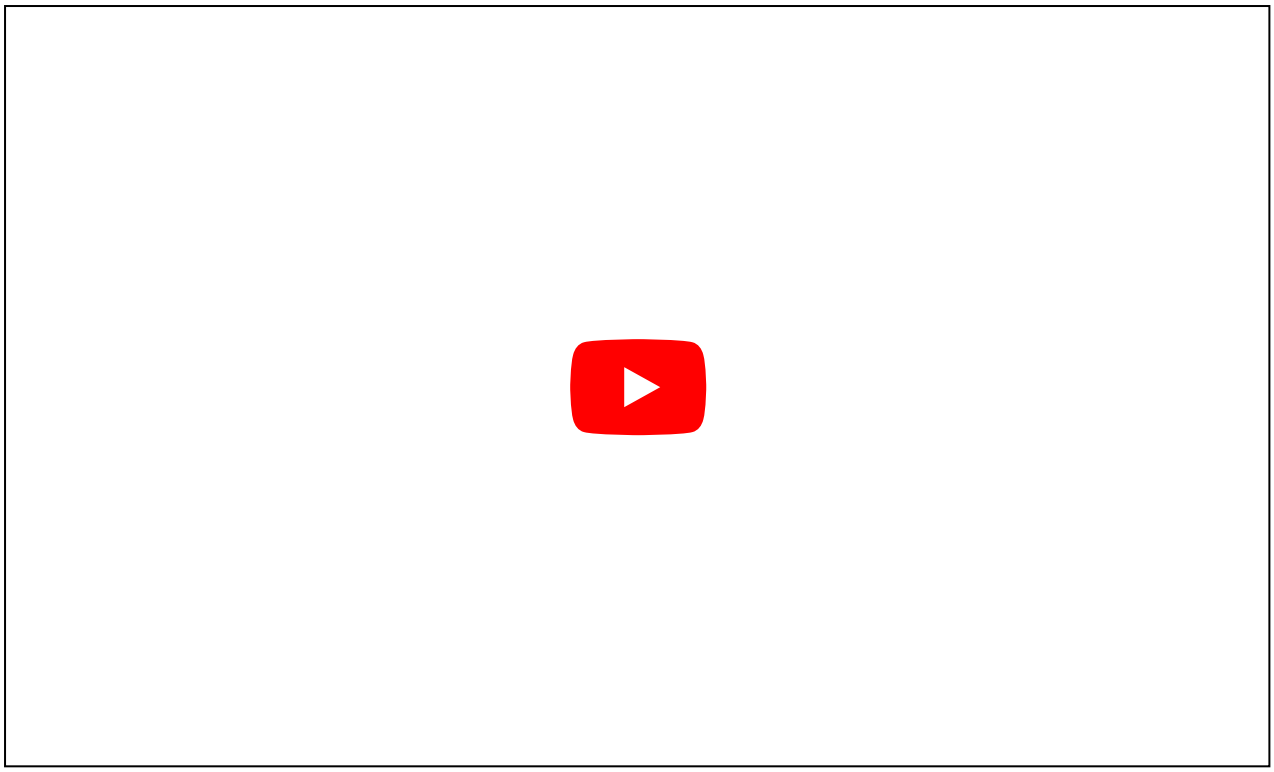
Learning Objectives

- Know the term perceptual blindness, and can learn to recognize essential (and non-essential) information with practice.
- Know ways to comment code, and understand the importance of variable names on readability of your code.
- Know the concept print-debugging, and know how to search for errors in the source code by printing.

We've so far been practicing the fundamentals of the language, such as variables, conditionals, loops, and methods. Let's now move on to look at some of the factors affecting the understandability of programs, and how errors are found.

A Programmer Blind to Their Own Code

A programmer often becomes blind to their code. Let's familiarize ourselves with this effect with the aid of the short video below. Count how many times the white-shirted players pass the ball between each other. The video contains instructions in English.



There's something else that also happens in the video that may go unnoticed at first. This effect is known as perceptual blindness, and is explained by the fact that as we focus on a specific task, our brains tend to filter out information that is irrelevant to that task. However, we don't always know what information is, in fact, essential and what is not - an example of this being when we study. Concentrating on a specific part of a study exercise can lead to relevant information being filtered out.

Fortunately, applying oneself to a given task lessens the occurrence of perceptual blindness. In other words, practice develops one's ability to distinguish between relevant and irrelevant information.

One way in which perceptual blindness manifests itself in programming practice is when concentrating on a specific part of a program draws attention away from seemingly correct, yet erroneous parts. For instance, while inspecting the correctness of a program's output, a programmer may fixate on the print statements, and mistakenly neglect some aspects of the logic.

Likewise, a programmer may focus on the most complicated aspect of a program featuring a loop, when in fact the error lies somewhere else completely. An example of this is the program below, which is used to calculate the average of user-inputted values. It contains an error, and when searching for it, the loop is typically the first target of focus.

```java
Scanner scanner = new Scanner(System.in);
int values = 0;
int sum = 0;

while (true) {
    System.out.println("Provide a value, a negative value ends the program");
    int value = Integer.valueOf(scanner.nextLine());
    if (value < 0) {
        break;
    }

    values = values + 1;
    sum = sum + value;
}

if (sum == 0) {
    System.out.println("The average of the values could not be calculated.");
} else {
    System.out.println("Average of values: " + (1.0 * sum / values));
}
```

Quiz:
Squish a bug

Points:
1/1

A bug still happens to remain in the program above. What kind of input can be used to find it? In the input options below, the character '\n' corresponds to a new line that is achieved by pressing the enter key.

Select the correct answer

-1\n

6\n2\n-1\n

✓ 0\n-1\n

The answer is correct

Answered Tries remaining: 1

Perceptual blindness is something that one cannot be eliminated completely. However, there are ways by which a programmer can lessen its effect - the first one being taking breaks, which requires that work is begun early. Code comments, proper naming of things, and "debugging" prints are additional examples of things that are also helpful.

Commenting the Source Code

Comments have many purposes, and one of them is explaining how the code works to oneself when searching for bugs. The execution of a relatively simple program is described below through the use of comments.

```
/*
 Prints the numbers from ten to one.
Each number is printed on its own line.
*/

// We create an integer variable named value,
// assigning the value 10 to it.
int value = 10;

// The loop execution continues until
// the value of the variable named value is less than or equal to
// zero. The excution doesn't stop _immediately_ when the value zero
// is assigned to the variable, but only when the condition of the
// loop is evaluated the following time.
// This always happens after the loop has executed
while (value > 0) {
    // we print out the value in the variable and a new line
    System.out.println(value);
    // we decrement the value by one
    value = value - 1;
}
```

Comments have no impact on the execution of the program, i.e., the program works in the same way with the comments as it does without them.

The comment style displayed above that is intended for learning purposes is, however, too elaborate for real development, where the goal is for the source code to be **self documenting**. This means that the functionality of the program should be evident from the way classes, methods, and variables are named.

The example can be "commented out" by encapsulating the code into an appropriately named method. Below are two examples of methods that do this - one of the methods is more general in its purpose compared to the other. The more general method assumes, however, that the user knows which of the two parameters is assigned the higher value and which the lower.

```
public static void printValuesFromTenToOne() {
    int value = 10;
    while (value > 0) {
        System.out.println(value);
        value = value - 1;
    }
}
```

```
public static void printValuesFromLargestToSmallest(int start, int end) {
    while (start >= end) {
        System.out.println(start);
        start = start - 1;
    }
}
```

Searching for Errors with Print Debugging

One required skill in programming is the ability to test and debug when searching for errors. The simplest way to search for errors is to use so-called print debugging, which in practice involves adding messages to certain lines of code. These messages are used to follow the flow of the

program's execution, and can also contain values of variables that live in the program.

Let's inspect the program already familiar to us from the previous question that was used to calculate the average of non-negative values.

```
Scanner scanner = new Scanner(System.in);
int values = 0;
int sum = 0;

while (true) {
    System.out.println("Provide a value, a negative value ends the program");
    int value = Integer.valueOf(scanner.nextLine());
    if (value < 0) {
        break;
    }

    values = values + 1;
    sum = sum + value;
}

if (sum == 0) {
    System.out.println("The average of the values could not be calculated.");
} else {
    System.out.println("Average of values: " + (1.0 * sum / values));
}
```

Had the program contained an error, print debugging could have been used to unravel its functionality by adding print statements in the appropriate places. The example below contains one possible example of a program containing print-debug statements.

```
Scanner scanner = new Scanner(System.in);
int values = 0;
int sum = 0;

while (true) {
    System.out.println("-- values: " + values + ", sum: " + sum);

    System.out.println("Provide a value, a negative value ends the program");
    int value = Integer.valueOf(scanner.nextLine());
    if (value < 0) {
        System.out.println("-- value is negative, exiting loop");
        break;
    }
```

```
    values = values + 1;
    sum = sum + value;
}

System.out.println("-- loop exited");
System.out.println("-- values: " + values + ", sum: " + sum);

if (sum == 0) {
    System.out.println("The average of the values could not be calculated.");
} else {
    System.out.println("Average of values: " + (1.0 * sum / values));
}
```

When a program is executed multiple times with appropriate inputs the hidden error is often found. Coming up with relevant inputs is a skill in its own right. It's essential to test the so-called corner cases, i.e., circumstances where the program execution could be exceptional. An example scenario would be one where the user does not enter a single acceptable value or enters zeros or very large values.

Quiz:

Burger chain app -- what does the program do?

Points: 1/1

A friend of yours was hired to create an application for a new burger restaurant chain that arrived in town. It should calculate whether a customer should be given a gift card. The intention is that after the first 1000 customers, every 25th should receive a gift card. In addition, every 2000th customer gets a larger gift card. Others receive nothing (expect for the chance to buy the restaurant's servings, of course!).

Currently, the code looks like this:

```
Scanner x = new Scanner(System.in);
System.out.print("Customer number: ");
int y = Integer.valueOf(x.nextLine());

if (y >= 1000 && y % 25 == 0) {
```

```
    System.out.println("Gets a gift card!");
} else if (y % 2000 == 0) {
    System.out.println("Gets a large gift card!");
} else {
    System.out.println("Gets nothing.");
}
```

There are a few issues with it. Explain what problems exist in the code and what causes them. Also give suggestions to fix those.

Your answer should be at least 20 words

Your answer

the customer count validation is incorrect. Past 2000 customers needs correction. sdas sdfgasdf sdfasf sdf sdf sdf sdaf sdf sdf

Words: 20

Answered The number of tries is not limited

You have reached the end of this section! Continue to the next section:

→ 2. Lists

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

3. Arrays

4. Using strings

5. Summary

Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI