Syntax highlighting effects on Java code readability

WILLIAM AGNÉR
HENRIK JOLÉRUS
Syntax highlighting effects on Java code readability

WILLIAM AGNÉR
HENRIK JOLÉRUS

Degree project, first cycle (15 hp)
Date: June 23, 2021
Supervisor: Richard Glassey
Examiner: Pawel Herman
School of Electrical Engineering and Computer Science
Swedish title: Läslighet av Javakod med hjälp av syntaxmarkering
Abstract

The readability of Java programs is important for both cost-effectiveness in Java development and for easing the process of learning and teaching Java programming. Syntax highlighting is used as a tool for increasing code readability through coloring and stylizing parts of code.

In this study, 12 participating computer science students (all familiar with Java) were asked to solve 10 Java problems. Some presented problems had syntax highlighting and some not. The problems were basic problems that all focused on retrieving information from code. This problem design was a distinguishing factor of this study in contrast to previous work. The experiments were measured in terms of problem-solving time, the number of correct answers, and eye movements.

Results showed, when averaging across all problems and participants, that syntax highlighting had a better average of correct answers by a margin of over 10%. Further, 7 out of 10 problems were completed quicker on average when presented with syntax highlighting, and surveys showed that participants found the problems with syntax highlighting easier to read. However, the results were very inconsistent across problems. In some instances, the fraction of correctness and problem-solving times were better when not using syntax highlighting.

The eye-tracking data did not provide much explanation for the results but showed in some instances that syntax highlighting reduced fixations in complex parts of code.
Sammanfattning

Läsbarhet av Javakod är viktigt, både för kostnadseffektivitet i mjukvaruutveckling och för att förenkla inlärning och utlärning av Javaprogrammering. Syntaxmarkering är ett verktyg för att öka läsbarhet av kod genom färgnings och dekoration av kodsegment.


Resultaten visade, i snitt över alla problem och deltagare, att syntaxmarkerade problem hade en högre andel korrekta svar med en marginal på över 10%. Vidare löstes 7 av 10 problem snabbare i genomsnitt när de presenterades med syntaxmarkering, och enkäter visade att de flesta deltagare tyckte att problemen med syntaxmarkering var lättare att läsa. Däremot var resultaten mycket inkonsekventa beroende på vilket problem man observerar. I vissa exempel hade icke syntaxmarkerade problem en högre andel korrekta svar och bättre problemlösningstider.

Ögonrörelsedatat gav inte mycket förklaring till resultaten, men visade i vissa fall att syntaxmarkering reducerade antalet fixeringar i komplexa kodsegment.
# Contents

1 Introduction ........................................... 1
   1.1 Motivation of study .................................. 1
      1.1.1 Relevance ........................................ 1
      1.1.2 The syntax highlighting tool ...................... 1
   1.2 Study concept ........................................ 2
      1.2.1 Objective .......................................... 2
      1.2.2 Scope .............................................. 3
      1.2.3 Measurements ...................................... 3
      1.2.4 Research question ................................. 3

2 Background ............................................. 4
   2.1 Programming in Java .................................. 4
   2.2 Importance of code readability ....................... 5
      2.2.1 In the industry ................................. 5
      2.2.2 Teaching programming ......................... 5
      2.2.3 Teaching object-oriented programming .......... 5
   2.3 Syntax highlighting ................................ 6
      2.3.1 Previous work .................................... 7

3 Methods ................................................. 9
   3.1 The experiment ...................................... 9
      3.1.1 Participants ..................................... 9
      3.1.2 Experiment process ............................ 9
      3.1.3 Problem design ................................ 10
      3.1.4 Eye tracking .................................... 10
      3.1.5 Experiment software .......................... 10
      3.1.6 Surveys ....................................... 11
   3.2 Data analysis ....................................... 11
      3.2.1 Task completion times and answers ............ 11
3.2.2 Eye tracking-data .................................. 11
3.3 Limitations ............................................ 12

4 Results .................................................. 13
4.1 Participant data ....................................... 13
  4.1.1 Pre-study survey ................................. 13
  4.1.2 Post-study survey .............................. 16
4.2 Problem solving data ................................. 18
  4.2.1 Correct answers ................................. 18
  4.2.2 Time-to-solve .................................... 20
4.3 Eye-tracking data .................................... 23

5 Discussion ............................................. 25
5.1 Data quality ......................................... 25
  5.1.1 Participant viability ............................ 25
  5.1.2 Experiment process integrity ............... 26
5.2 Results .............................................. 26
  5.2.1 Correct answer percentage .................. 26
  5.2.2 Problem-solving times ....................... 27
5.3 Limitations .......................................... 28
  5.3.1 Participants .................................... 28
  5.3.2 Problem design ................................ 29
  5.3.3 Eye tracking data ............................. 29
  5.3.4 Experiment setup ............................. 29

6 Conclusions ........................................... 30
  6.1 Future work ....................................... 30

Bibliography .............................................. 31

A Questionnaires ........................................ 34

B Heatmaps ................................................ 39
Chapter 1

Introduction

1.1 Motivation of study

1.1.1 Relevance

As computers become relevant to more and more aspects of society, programming has grown as a topic of study. Programming (the writing of computer programs) is now part of curriculums in many different areas of university studies such as science, technology, economics, and medicine. Programming has also expanded to high school level curriculums [1].

The area of teaching such introductory programming studies - i.e. teaching programming to those who have not written programs before - has been a widely studied subject. This area of study involves such things as solving questions regarding which programming language is most appropriate for learning how to write programs or the question of how to explain a certain concept of programming to a given target group [2].

1.1.2 The syntax highlighting tool

Another part of teaching programming that is being explored is the use of tools for making programming more approachable. These tools usually take the form of software and are, by appearance, made to provide a more stimulating programming experience. For example, programs used to write code are often decorated with colors, buttons, and animations to attract users and ease the threshold for new programmers [3].

Syntax highlighting would be considered by many to be included in this class of stimulating programming decoration tools. In fact, in contrast to most
other decoration tools, one could argue that syntax highlighting actually could measurably improve productivity.

Syntax highlighting (see listings 1 and 2) entails coloring or stylizing classes of key symbols and words (specifically; lexical tokens) often as part of code editing software. The goal of this is to make it easier to read the code, since - hypothetically - it is faster for a programmer to find errors or information about the code through color than through code context [4].

```java
int a = "Hello world!";
System.out.println(a);
```

Listing 1: Without syntax highlighting

```java
int a = "Hello world!";
System.out.println(a);
```

Listing 2: With syntax highlighting

Hence, in theory, syntax highlighting can provide a less frustrating, more stimulating, and more efficient programming process. This is useful since it hypothetically can attract more people to the programming job market, which is currently in deficit. It can also help current programmers find their job or their studies more stimulating, and it can help them be more efficient.

### 1.2 Study concept

#### 1.2.1 Objective

This study aims to build upon previous research regarding the effectiveness of syntax highlighting, while significantly reducing scope in hope that this will render useful results. Previous work has been focused on solving full programming problems and/or analyzing a subjectively improved programming experience.

The study will revolve around in-person programming experiments, as it appears as the most intuitive way of investigating the effect of a tool on a given human process. This is the same method as used in previous work. The experiments will, however, be focused on one part of solving programming problems instead of the full process of solving programming problems. This is because the process of solving a full programming problem can consist of a range of steps, adding to the variability and inconsistency of results.

It was decided that *information retrieval from code* was the most useful problem-solving step to investigate, as it is a relatively uncomplicated step and one where elements of interest in code can be spread out visually. This is useful since this study will follow the methodology of previous work in
utilizing eye-tracking for further insight, and spreading out elements of interest can accommodate eye-tracking data that is easier to analyze.

1.2.2 Scope

As a result of the aim of the study, as well as availability and time constraints, the experiment participant group will consist of a relatively small group of university students studying computer science. It will focus on Java programming, since it is the most popular introductory programming language [5], and has been used by individuals expected to participate in the study. Furthermore, the study will only focus on one syntax highlighting theme.

1.2.3 Measurements

The most important measurements of the study are the success rates for the problems (or fraction of correct answers), and time spent on problem completion. The effect that syntax highlighting has on these measurements will be analyzed. The study will also utilize eye-tracking to investigate possible explanations for the results of the measurements.

1.2.4 Research question

Given the scope and objective of the study, the research question was formulated as follows:

"Can university students studying programming retrieve information quicker from Java code, as a consequence of more efficient identification of syntax elements, with the help of a well-renowned syntax highlighting system?"
Chapter 2

Background

2.1 Programming in Java

Java is part of a commonly used programming paradigm known as object-oriented programming. Just as with a majority of programming languages, the written code of an object-oriented program can be read like a list of instructions for the computer to perform in order to change the state of the program or the state of the computer it runs on [6]. An example of Java code can be seen in Listing 3.

```java
public class Person {
    String name;
    int age;
}

public class Student extends Person {
    public static void main(String[] args) {
        Student student = new Student();
        student.name = "John Doe";
        student.age = 22;
        System.out.println(student.name);
    }
}
```

Listing 3: An example of Java code

Object-oriented programming additionally involves the creation and management of objects, which play a similar role to variables in a mathematical function, but can hold many instances of data at once (fields in Java) as well as procedures (called methods in Java). Classes, which operate as templates for objects, can inherit fields and methods from other classes hierarchically [6].

In order to comprehend what a Java program does, a programmer can read the code from start to finish. If the programmer just wants to find certain
information about what happens in the program, he/she can look for certain keywords in it [7].

### 2.2 Importance of code readability

#### 2.2.1 In the industry

Code can be written and presented in a variety of ways, where some ways are easier to comprehend than others. Having code that is easy to comprehend and follow, in other words; making it more readable, is part of making code maintainable. This is because it reduces the risk of introducing bugs when code is shared between developers [8]. Being able to maintain code in an effective manner can have positive impacts on cost-effectivity, as software maintenance can cost as much as nine times the cost of all other costs related to development[9].

#### 2.2.2 Teaching programming

Code readability is also a factor in teaching programming. One study stated that programming is a difficult task to teach and to learn. Since many students find programming difficult, this could have consequences for their development as software engineers. A couple of aspects were proposed as effective tools regarding teaching and learning programming. These included active learning, mechanisms to help students analyze their metacognitive and independent learning skills, and visualizing concepts in different ways using tools. They also stated that many programming novices can understand the syntax elements of individual components, but have a harder time piecing this together into a functioning program [10]. Another study found that using learning tools to visualize algorithms as a way of teaching programming was effective, and how the students used the tools was even more important [11].

#### 2.2.3 Teaching object-oriented programming

How to teach object-oriented programming specifically has also been researched and debated. One study made sure to state how important the introductory course in programming is to students’ future coding habits and their ability to educate themselves further. The hardest aspects of teaching Java to first-year students were grasping concepts such as variable scope errors, integer arithmetic errors, operator precedence problems, and casting objects. To teach Java
more effectively, several changes were incorporated into the teaching method. Among these the most effective were familiarizing students with an integrated development environment, addressing parts of the Java programming language that were especially difficult to grasp, and encouraging active learning and self-assessment [12].

2.3 Syntax highlighting

Expanding on ways to improve code readability, theoretically, is the concept of syntax highlighting. In programming, the concept of syntax highlighting is the coloring or stylizing of certain key parts of code.

The coloring is often done in the context of the program as a whole and provides information that the programmer otherwise would have had to find out by reading a greater portion of the program [4].

Examples of style changes applied to syntactical elements are:

- Coloring string literals in green to highlight start- and endpoints
- Highlighting class fields in purple to make a distinction between fields and variables
- Coloring unused methods and variables in gray

A common way of applying syntax highlighting is to make use of syntax highlighting tools that are available in software used for programming. These tools automatically and continuously apply syntax highlighting to code as the programmer writes the program. An example of code with syntax highlighting can be seen in the Java code in listing 4, where syntax highlighting has been applied to the example code from listing 3:
public class Person {
    String name;
    int age;
}

public class Student extends Person {
    public static void main(String[] args){
        Student student = new Student();
        student.name = "John Doe";
        student.age = 22;
        System.out.println(student.name);
    }
}

Listing 4: An example of Java code with syntax highlighting

Syntax highlighting is applied on the premise that it makes it easier to identify the semantics of the code, thereby streamlining the comprehension process for the programmer. Some studies have indicated that this presumptive streamlining effect is real [13] [14], which will be discussed further in the section on previous work.

Syntax highlighting themes

In the above example, the text string literal "Hello world!" can be seen as highlighted in red, the fields being accessed on rows 8, 9, and 10 can be seen in green, and the types "int" and "void" can be seen in bold red. All of these stylistic changes have been made as part of syntax highlighting.

The stylistic changes that are applied are a matter of preference. A certain system or group of syntax highlighting changes is commonly referred to as a syntax highlighting theme, and software that enables syntax highlighting can provide multiple themes for the user to choose from.

For Java code, it is safe to assume that the most common syntax highlighting theme for programming students is the default theme for IntelliJ, the most popular integrated development environment for Java programming among future developers [15]. This syntax highlighting theme is known as "Darcula". This theme will be the only theme used in this study.

2.3.1 Previous work

The question of whether this hypothetical benefit of syntax highlighting actually exists has been explored to some extent. In one study, researchers Tanya
Beelders and Jean-Pierre L. du Plessis investigated whether student programmers experience more difficulty in reading and comprehending source code when it is presented without syntax highlighting. The study mostly relied on survey scores regarding students’ experiences of reading a set of programs presented with- and without syntax highlighting, and the study found that the highlighted programs scored better in readability and appearance [14].

Another study, published in *Empirical Software Engineering*, measured the correctness in which novice programmers could solve small programming tasks with the help of syntax highlighting. The data from the study provided no evidence that syntax highlighting improves novices’ ability to comprehend source code [16].

Furthermore, one study by Advait Sarkar used the same method of having participants solve programming tasks. In this study, the conclusion was that syntax highlighting significantly improved task completion time [13].

One commonality between the study by Beelders/L. du Plessis and the study by Sarkar is the use of eye-tracking technology to gain more insight into their experiments. However - in both instances - the eye-tracking data was not insightful enough to be of much use in drawing conclusions.
Chapter 3

Methods

3.1 The experiment

3.1.1 Participants

Participants consisted of computer science students studying at KTH in Stockholm. This group of participants was chosen primarily due to availability and relevance to the research question. The research question specified students working on Java code. Most computer science students at KTH were familiar with Java to some degree. Also, most students had completed courses DD1337 and DD1338. Completion of these courses gave some assurance that participants can:

- solve computational problems by means of programming,
- analyze, choose, use and implement basic algorithms and data structures as described in the intended learning outcomes of the course DD1338 [17].

3.1.2 Experiment process

Firstly, participants were asked to fill a pre-study survey. The participants were then informed of the challenge of the experiment, being that the participant reads a snippet of Java code and is asked to type in a text field what the output of the program is. This challenge was repeated 10 times for 10 different code snippets and the participant’s eye movements were recorded during each challenge. The experiment was complete once all 10 challenges were completed and the participant completed a post-study survey.
3.1.3 Problem design

The problems were designed with the research question in mind. Specifically, the output of each code snippet was determined by some other part of the code, making the participant look to other parts of the code to retrieve the information necessary to answer the question. Further, certain irrelevant parts were added to the code snippets on the premise that irrelevant parts are easier to ignore - theoretically - when using syntax highlighting.

The finalized code snippets can be seen as part of the results chapter. The snippets were divided into two sets, A and B, both consisting of the same 10 code snippets but with some snippets having syntax highlighting (S) and some not (N). The highlighted snippets were then distributed in the following pattern for set A: S, N, S, N, S, N, S, N, S, N and in the following pattern for set B: N, S, N, S, N, S, N, S, N, S. Half of the participants were presented with set A, while the other half of the participants were presented with set B.

As previously explained, "Darcula" is a popular syntax highlighting theme, and was used for code snippets that had syntax highlighting. Code snippets without syntax highlighting were simply displayed as white text on a black background instead.

3.1.4 Eye tracking

The Tobii Eye Tracker 4C was the hardware used for gathering eye movement data. During the experiment, the gaze position of the participant was captured 50 times per second. For each new participant, and before they were introduced to the code snippets, a calibration for the eye-tracking hardware was conducted to achieve more accurate results.

3.1.5 Experiment software

Unity was used as a platform to conduct the experiment as Unity allowed easy creation of simulation environments. An environment was created where the previously described code snippets were displayed after one another for each participant along with a text field for writing answers. The Unity environment included a program that communicated with the eye-tracking hardware and translated eye tracking data during the experiment to pixel-described gaze coordinates in an output file.
3.1.6 Surveys

The pre-experiment survey served as a means of obtaining data and screening participants on programming experience, and proficiency in the Java programming language. It also asked about participant’s preconceived ideas on syntax highlighting. After participants had completed the experiment, they were asked to fill out a post-experiment survey. The post-experiment survey was used to determine whether the experiment had been completed as expected. The surveys can be found in appendix A.

3.2 Data analysis

3.2.1 Task completion times and answers

The completion times for each snippet and participant were recorded as part of the experiment software and were aggregated on a spreadsheet by code snippet and by whether or not they had syntax highlighting. Along with completion times, the answers to each snippet were recorded in the same fashion. Column charts and box plots were then created to visualize the data. The spreadsheets and charts were managed using Microsoft Excel software.

3.2.2 Eye tracking-data

As mentioned, the eye-tracking data was captured 50 times per second and was recorded as rows in an output file for each problem and participant. Software for analyzing this gaze data over was found and was used during the data analysis process. The software used for analyzing the gaze data was the toolkit PyGaze [18].

Fixations and heatmaps

Since eye-tracking data was large and difficult to analyze, the data was processed into fixations using PyGaze. Fixations are circles of pixel coordinates on the monitor within which the participant leaves their gaze for a certain amount of time. The fixations can in turn be used to generate visual analytics graphs such as heatmaps, also using PyGaze. In a heatmap, parts of the displayed image are colored, with the color depending on the level of fixations in that area.
3.3 Limitations

Some limitations of the experiments were realized before execution. The eye-tracking hardware appeared to have a rather large margin of error and was found to not be suitable for scientific use. Further, the small target group of participants required data to be uniform and conclusive in order to draw any proper conclusions, and such conclusions would be limited regardless. The COVID-19 pandemic might have affected the number of individuals willing to participate.
Chapter 4

Results

This chapter will display data gathered in the study. 12 university students participated. The experiments took place on May 4th and May 5th of 2021.

4.1 Participant data

4.1.1 Pre-study survey

As mentioned, the pre-study survey was filled out by participants before participants were introduced to the experiment and the problem set. This was done to ensure that participants were part of the target group of the study. The results are presented in the following charts. Results from some survey questions are excluded as they were deemed irrelevant to the examination of participant viability. Figure 4.1 shows the number of participants who had Java as one of their main languages. Figure 4.2 further illustrates how many participants had completed the courses DD1337 and DD1338. Figure 4.3 presents the total estimated programming experience in years for the participants. The fraction of participants who had tried using the syntax highlighting theme used in the experiment before is seen in figure 4.4.
A majority of participants mentioned Java when asked about their main programming languages.

All participants had completed courses DD1337 and DD1338 at KTH.
When asked about their programming experience, most participants answered that they had between three and five years of programming experience.

Three quarters of participants had previously tried using the syntax highlighting theme used in the experiment.

Figure 4.3: Total estimated programming experience in years for participants.

Figure 4.4: Participants who had used the "Darcula" syntax highlighting theme as part of the IntelliJ IDEA.
4.1.2 Post-study survey

Further verifying data quality, a post-study survey was filled out by participants after their respective experiments were completed. The main intention behind these surveys was to ensure that no major distractions occurred that may have corrupted experiment data, as well as to gauge the programming experience with- and without syntax highlighting. The results are presented in the following charts. Figure 4.5. shows the level of confidence participants had in their answers. Figure 4.6 presents the number of participants that thought the exercises with syntax highlighting were easier to understand. Figure 4.7 illustrates how many participants felt that they could focus on the task during the experiment.

80% of participants felt confident in their answers.

![Figure 4.5: Level of confidence participants had in their answers.](image-url)
A substantial majority found that exercises with syntax highlighting were easier to understand.

All participants answered that they could focus on the task during the experiment.
4.2 Problem solving data

4.2.1 Correct answers

The answers given by each participant were recorded and compared to the correct answers. These results were then grouped for each problem and by whether or not the code had been presented to the participant with- or without syntax highlighting. All in all, each problem was attempted 6 times with syntax highlighting and 6 times without syntax highlighting. The fractions of correct answers for each problem are presented in figure 4.8. The number of problems where each problem group had better results are displayed in figure 4.9, and the average correctness across all problems can be seen in figure 4.10.

![Percentage of correct answers](image)

Figure 4.8: Percentage of correct answers for each problem, with and without syntax highlighting.

From figure 4.8, it is seen that most problems had a better average correctness when displayed with syntax highlighting. This fraction of problems is displayed in 4.9.
Chapter 4. Results

Figure 4.9: Number of problems with highest percentage of correct answers, with and without syntax highlighting.

Averaging over all problems, it can be seen that problems presented with syntax highlighting rendered a greater degree of correct answers by a margin of over 10%.

Figure 4.10: Average correct answer percentage across all problems, with and without syntax highlighting.
4.2.2 Time-to-solve

Furthermore, the time between a problem being displayed until an answer was given, was also recorded. As with the rate of correct answers, this data was grouped by syntax highlighting to facilitate analysis of the effect of syntax highlighting. The amount of time to solve each problem is illustrated in figures 4.11, 4.12 and 4.13.

![Average problem completion time](image)

Figure 4.11: Average problem completion time for each problem, with and without syntax highlighting.
From figure 4.11, it is seen that most problems had a better average completion time when displayed with syntax highlighting. This fraction of problems is displayed in 4.12.

Figure 4.12: Number of problems with lowest completion times, with and without syntax highlighting.
Figure 4.13 shows that completion times were somewhat lower in general, when displaying all completion times in a boxplot.

Figure 4.13: All completion times for all problems, with and without syntax highlighting.
4.3 Eye-tracking data

The eye-tracking data was analyzed as explained in the methods section.

The fixation heatmaps were created by combining eye-tracking data from every test subject for each problem. The result was 20 different heatmaps. Two for each problem, one from set A, and one from set B. In other words, two per code snippet with one using syntax highlighting and one in black-and-white. These two heatmaps for each problem are presented in Appendix B, and an example of these heatmaps can be found in figure 4.14. The participants looked for a longer period of time at sections with more intensely red coloring. In this example - for the code presented with syntax highlighting - it is evident that the participants had fixed their gaze on the main method the most, while the "setType" method had no fixations at all. All resulting heatmaps can be found in appendix B.
Figure 4.14: Example of fixation heatmaps.
Chapter 5

Discussion

This chapter will discuss the findings of the result chapter, how they can be interpreted as well as post-experiment realizations regarding limitations.

5.1 Data quality

Questions in pre-study and post-study surveys were asked to confirm the viability of participants and to investigate the quality of their experiment experience. This data will be explored in this section.

5.1.1 Participant viability

Recollecting the scope of the study, the target group for the experiments was students studying computer science at a university level with Java programming experience.

The most insight garnered regarding these criteria comes from the fact that all participants had completed courses DD1337 and DD1338 at KTH, as seen in Figure 4.2. This means that all participants beyond a reasonable doubt possess the skills necessary to complete the experiment properly, as explained in section 3.1.1. Furthermore, a majority of participants mentioned Java when listing their main programming languages (Figure 4.1), and all participants estimated their programming experience to be somewhere between 1-5 years (Figure 4.3).
5.1.2 Experiment process integrity

Observing data from post-experiment surveys, all 12 participants answered that they could focus on the task during the experiment as seen in Figure 4.7. This by itself indicates that none of the participants were distracted, which in turn means that the data can be used without discarding data from any specific participant as a result of distractions and thereby uninsightful data. Further, most participants (10 out of 12) answered a 4 out of 5 when asked to rate their confidence in their problem answers (Figure 4.5).

5.2 Results

5.2.1 Correct answer percentage

Using syntax highlighting to retrieve information quicker from code - as set out to be explored at the conception of this study - invariably implies exploring the effectiveness and correctness of such information retrieval. This is why data regarding the correctness of answers given throughout the experiments of this study is of interest.

Figure 4.10 shows that, if averaged over the full problem set, participants with syntax highlighting answered correctly to a greater degree than students who did not. Specifically, answers given with syntax highlighting had higher average correctness by a margin of over 10%.

The fraction of correct answers varied across problems. This can be observed in figures 4.9 and 4.8. Problems 1, 3, 5, 7, 8, and 9 were correctly answered to a greater degree when participants used syntax highlighting, while the opposite was true for problems 2, 4, and 10.

This variability of the degree of correctness brings uncertainty to these results. It casts doubt on the intuitive conclusion that syntax highlighting helped participants in retrieving correct information from the code, because of the overall higher fraction of correct answers. Perhaps the variability would have been reduced had the problems been more similar. In other words, narrowing down problem types to further focus in on one particular problem-solving step may have reduced this inconclusiveness.

It is clear from Figure 4.8 that syntax highlighting had a similar grade of effect on problems 1, 3, 7, 8, and 9. An apparent similarity between a majority of these problems is that they involved for loops and if statements, in contrast to other problems where the problem-solving process was more focused on dissecting inheritance structures and retrieving information stored in fields.
When observing the eye tracking data for these problems (Figure B.1, B.3, B.7, B.8, B.9), somewhat of a pattern emerges. Focus points for fixations were, in general, quite different between code snippets with syntax highlighting and without.

For instance, in the case of problem 1 with no syntax highlighting, fixations are far more focused on the \textit{if} statement in the heatmap. This indicates that participants spent more time and exerted more mental strain when analyzing the if statement when they did not have the assistance of syntax highlighting. This can be explained by the increased visual contrast caused by the color decorations applied by the syntax highlighting theme, helping participants distinguish between syntax tokens (variables and keywords in this example).

For problem 3, several lines of code were irrelevant as they edited a field of the \textit{Bus} object that was never used. These irrelevant lines of code have a great density of fixations in the heatmap with no syntax highlighting, while the heatmap for syntax highlighting is less focused on these irrelevant parts. One may assume the code snippet with syntax highlighting allows for the \textit{maker} field to be more easily spotted as the one being accessed since it is highlighted in purple.

Looking at other problems where syntax highlighting had an effect on the percentage of correct answers, like problems 7 and 9, the results are not as telling. This is because the fixations do not have any particular focus points and do not differ to any significant degree between the two heatmaps.

\section{Problem-solving times}

In the context of the goal of the study, the most crucial information from the study is the data regarding the problem-solving times for the participants. This is because the research question revolves around whether syntax highlighting results in "quicker" information retrieval from Java code, directly referring to the time it takes to retrieve information from the code (which in this case means 'solving the problem').

As with the data regarding correct answers, the effect of syntax highlighting on problem-solving times varied greatly between different problems (as seen in figure 4.11). In some cases, problem-solving times were lower (better) for code snippets without syntax highlighting. This, again, leaves little room to make any direct conclusions or assume any pattern.

When examining the time data in a broader sense, however, code snippets with syntax highlighting appears to have been solved quicker than those without. In Figure 4.13, one can see that the upper quartile and mean time to solve
a problem was lower for code snippets with syntax highlighting. Furthermore, Figure 4.12 shows that a majority of problems were completed quicker when using syntax highlighting.

As mentioned, these general results regarding the effects on information retrieval times are difficult to draw conclusions from, especially considering the variability between problems. However, when observing that - in a general sense - problems were solved more accurately when using syntax highlighting combined with the fact that they also were solved quicker, this does weigh in favor of the original hypothesis that syntax highlighting actually affects information retrieval from code positively.

On the other hand, further inconclusiveness is found when comparing problems deviating from the hypothesis between answer correctness data and time data. The problems where non-syntax highlighted code snippets rendered more correct answers were not the same as the non-syntax highlighted code snippets that were completed quicker than their syntax highlighting counterparts.

Furthermore, in terms of the time data specifically, the differences were very small. In fact, for problems 1, 2, 3 and 9, the relative differences were only 6.8%, 1.8%, 3.5%, and 5.4% respectively. In problems where the difference was greater, the eye-tracking data did not provide much insight. For example, the heatmaps for problem 4 (figure B.4) show that participants using syntax highlighting had more fixations in irrelevant parts (contrary to the hypothesis). For problems 6 and 7 (figures B.6 and B.7) there does not appear to be much difference at all between the two heatmaps.

5.3 Limitations

5.3.1 Participants

Possibly as a consequence of the fact that experiments took place amid social restrictions due to the COVID-19 pandemic, the fraction of students who accepted the offer to participate in the experiment was small. One can assume that a larger participant group would have contributed to more consistent results.

It was also realized that the conclusiveness of the study would have been helped by screening participants for colorblindness since different degrees of colorblindness speculatively can alter how syntax highlighting affects problem-solving.

Further, the variance in the amount of programming experience between
participants (seen in Figure 4.3) is a factor that may have affected results but was not accounted for.

5.3.2 Problem design

Seeing as how this study aimed to focus on one problem-solving step and limit the variability of the data rendered from the problem-solving instances, it is clear that further emphasis on such reductions of variability would have been necessary. This is due to the lack of clarity from the data regarding correct answers and time-to-solve as well as eye-tracking data.

In relation to the design of the problems, the lack of emphasis on specifically investigating the information retrieval step is noticeable in the problem-solving times being higher than expected. Speculatively, the lengthy solving times adds significant variability in the solving times.

5.3.3 Eye tracking data

The greater-than-expected solving times also contributed to adding substantially more eye-tracking data, making the eye-tracking data more difficult to analyze. In some cases, heatmaps were completely uninsightful due to confusion in fixation data (such as Figure B.10)

Further, before the study, many assumptions were made regarding how the eye-tracking data could be analyzed. For instance, it was assumed that the fixations could be tracked sequentially across the display. This proved to be infeasible considering the large amounts of gaze movements that occurred as a result of the magnitude of the code problems that were presented.

Another unforeseen problem was that many fixations were focused on the answer input field, which possibly negatively affected the insightfulness of the fixation heatmaps.

5.3.4 Experiment setup

As the eye tracker used for this study is not intended for scientific purposes, it is difficult to gauge the accuracy of the eye-tracking data. Further, lighting conditions may have varied throughout experiments as daylight was the main source of lighting.
Chapter 6

Conclusions

Confirming results of previous studies, a substantial majority of participants subjectively experienced the problems with syntax highlighting to be easier to understand than those without.

When studying the empirical data, code snippets with syntax highlighting had a higher fraction of correct answers by a margin of over 10% and a lower average completion time for seven out of ten problems.

This pattern was not obvious across all problems, however, and is not convincing enough to draw any proper conclusions from. Causality could not be determined with the use of eye-tracking data. It was, however, observed that comparatively, complex problem parts had a greater degree of gaze fixations when not using syntax highlighting in some instances.

6.1 Future work

Future studies of this kind should emphasize the information retrieval step discussed in this study further. In practice, this means presenting problems that are less complex and more similar to each other than the ones explored as part of this study.

A larger participant group would also, speculatively, lead to more consistent results.

Furthermore, if eye-tracking data is to be used, the process for collecting and - above all - analyzing such data must be thoroughly tested beforehand to ensure that it can provide any insight. The equipment and environment for collecting the data should be accurate and consistent.
Bibliography


[16] Christoph Hannebauer, Marc Hesenius, and Volker Gruhn. “Does synta-

tax highlighting help programming novices?” In: Empirical Software


10.1007/s10664-017-9579-0. url: https://doi.org/10.1007/s10664-017-9579-0.

[17] KTH. “Before choosing course DD1338 Algorithms and Data Struc-
tures 6.0 credits”. In: KTH Course and programme directory (2021).


cross-platform toolbox for minimal-effort programming of eyetracking

Appendix A

Questionnaires
Pre-study questionnaire
(As part of study: “Syntax highlighting effects on Java code readability”)

*Required

1. Have you completed courses DD1337 and DD1338 at KTH? (1st & 2nd INDA Java courses) *

   Mark only one oval.
   
   ☐ Yes
   ☐ No

2. What is your total estimated programming experience in years? *

   Mark only one oval.
   
   ☐ Less than one year
   ☐ One year
   ☐ Two years
   ☐ Three years
   ☐ Four years
   ☐ Five years
   ☐ More than five years

3. What is your main programming language? *


Figure A.1
4. What is your self-estimated skill in the Java programming language?

*Mark only one oval.*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>Professional</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Beginner

Syntax highlighting is the coloring of parts of code in order to improve readability. The code snippet below on the right uses syntax highlighting.

```java
int a = "Hello world!";
System.out.println(a);
```

5. I understand what "syntax highlighting" is.*

*Mark only one oval.*

☐ Yes

☐ No

6. I have tried using syntax highlighting: *

*Mark only one oval.*

☐ Yes

☐ No

Part 3: Syntax highlighting

7. I use syntax highlighting regularly:

*Mark only one oval.*

☐ Yes

☐ No
8. I have tried using the syntax highlighting theme “Darcuia” as part of the IntelliJ IDEA.

Mark only one oval.

☐ Yes  
☐ No  
☐ I don’t know

9. I believe that syntax highlighting...

Mark only one oval per row.

<table>
<thead>
<tr>
<th></th>
<th>Strongly agree</th>
<th>Agree</th>
<th>Disagree</th>
<th>Strongly disagree</th>
</tr>
</thead>
<tbody>
<tr>
<td>makes code easier to read</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>makes my programming more efficient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>makes programming less frustrating</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10. I believe that syntax highlighting only makes programming more efficient due to the placebo effect.

Mark only one oval.

☐ Yes  
☐ No

Figure A.3
Post-study questionnaire
(As part of study: “Syntax highlighting effects on Java code readability”)
*Required

1. Could you focus on the task during the experiment? *
   
   Mark only one oval.
   
   ☐ Yes
   ☐ No

2. Were the exercises with syntax highlighting easier to understand than the ones without syntax highlighting? *
   
   Mark only one oval.
   
   ☐ The ones with syntax highlighting were easier to read
   ☐ The ones without syntax highlighting were easier to read
   ☐ I did not notice a difference
   ☐ Other: __________________________

3. Did you feel confident in your answers to the exercises? *
   
   Mark only one oval.

   1  2  3  4  5

   Not very confident ☐ ☐ ☐ ☐ ☐ Very confident

Figure A.4
Appendix B

Heatmaps
Figure B.1: Heatmaps for code snippet 1.
Figure B.2: Heatmaps for code snippet 2.
Figure B.3: Heatmaps for code snippet 3.
Figure B.4: Heatmaps for code snippet 4.
Figure B.5: Heatmaps for code snippet 5.
Figure B.6: Heatmaps for code snippet 6.
Figure B.7: Heatmaps for code snippet 7.
Figure B.8: Heatmaps for code snippet 8.
Figure B.9: Heatmaps for code snippet 9.
Figure B.10: Heatmaps for code snippet 10.