On Efficiency and popularity of Java Lambda Expressions

Dillon Boatman
Department of Computer Science
Arkansas State University
Jonesboro, Arkansas, USA
dillon.boatman@smail.astate.edu

Amit Kathayat
Department of Computer Science
Arkansas State University
Jonesboro, Arkansas, USA
amit.kathayat@smail.astate.edu

Blake White
Department of Computer Science
Arkansas State University
Jonesboro, Arkansas, USA
blake.white2@smail.astate.edu

ABSTRACT
Lambda expressions were introduced into various existing programming languages, namely Java, to improve readability, as well as create a way to achieve a one method interface. In this paper we set out to analyze whether lambda expressions are actually being used in live code, as well as determining if lambda expressions exhibit a reduction in lines of code when used.

CCS Concepts
• Software and its engineering → Software notations and tools → General programming languages → Language features → Procedures Functions and subroutines

Keywords
Lambda Expression; Lines of Code; Efficiency

1. Introduction
Lambda expressions were introduced in the Java SE 8 framework. This type of expression is used to bridge the gap between object-oriented programming to functional programming. Prior to the advent of lambda expressions, every function in java had to belong to a class. The tie to functional programming is that this type of expression can mask as an object when passed as a parameter but be executed as a function. This opens the door to a lot of new possibilities, namely in Java.

The general syntax is:

$$(x1, x2) \rightarrow \{\, return\ x1 * x2 \};$$
(1)

where x1 and x2 are parameters and the arrow indicated the starting portion of the function body, which is in brackets. As for the type of the parameters, there are cases when you would need to specify type, but it is not always necessary. After introducing this type of expression, there is the question of why or if it is even necessary? To be exact, it isn't technically necessary to use lambdas. It is achievable to obtain the same results without lambda, but they can make a programmer's life easier at times.

One main benefit for using lambda expressions is simply the convenience it can provide once a programmer learns how to use them. If you need a simple function that you are possibly only going to use once, it is easy to declare a nameless function 'on the fly'. Especially in Java, you would be able to quickly insert a function without having to make a subsequent class for it. Lambda expressions can also be written inline, so in minimum cases it would simply be one line versus many. The next benefit is conditional based on your knowledge of these expressions: It can make your code easier to read. They are compact and without fluff. If you are familiar with the syntax, it can be easy to reason about what someone's code is doing.

Now to the negatives of using lambda expressions. This contradicts the last pro that I introduced; if you are fresh to programming, these can be a nightmare to read. It is a new syntax that does not appear to follow the same traditional rules of previous programs you may have written. They are also generally not part of a traditional college curriculum, so a programmer may only see them 'in the wild' in industry.

2. Background
In this study, we selected six different open-source projects to analyze. These projects were incredibly different in nature; they are projects that perform different functions. The one aspect they have in common is being comprised of mainly java code. In our view, this gives a greater expansion of the different uses of lambda expressions. The following table gives list of the project and functionality.

Table 1: Projects

Project Name	Use
Apache Solr	Enterprise Server
Arduino	Electronics Prototyping
Google Guava	Google Core Library's
Tomcat	WebSocket Technology
Eclipse Smart Home	Smart Home Solutions
Apache Flink	Distributed Processing

3. Related Works
In our project, we analyzed multiple research papers with the interest of lambda expressions and their effectiveness in different aspects of Java code.

3.1 Lucas et al.
In the research paper entitled, "Does the Introduction of Lambda Expressions Improve the Comprehension of Java Programs?", Lucas et al. execute multiple methods of qualitative and quantitative analysis

to decide whether this feature is a useful addition to the Java language. The two methods of analysis are using pre-written models and polling the community to determine the effectiveness of lambda expressions and code comprehension. Their formal research questions are "Does the use of lambda expressions improve program comprehension?", "Does the introduction of lambda expressions reduce source code complexity?", "What are the most suitable situations to refactor a code to introduce lambda expressions in Java?", and "How do practitioners and students evaluate the effect of introducing a lambda expression into a legacy code?".

For their quantitative research, they used Lines of Code (LOC) and Cyclomatic Complexity (CC) as their two deciding metrics. Alongside this, the use of the "Buse and Weimer" model and the "Posnett et al." Model. The Buse and Weimer model is used to create an estimate of the comprehensibility of a code snippet by using aspects like length of each individual line of code, as well as number and length of identifiers. The Posnett Model is a furthering of the previous model, in which they dive further using the number of lines of code, the volume of the code, and the entropy of the code.

For their qualitative research, they polled two different groups: Seasoned programmers with experience in Java programming and undergraduate students. The two groups were given four code snippets, where there were two that performed a task without the use of lambda expressions and the other two were the same snippet but refactored to use lambda expressions. They were then given three questions regarding the snippets. The first being if they agreed that the refactoring to use lambda expressions was beneficial, one a 1-5 scale. Then they were given a yes or no question, regarding if they preferred the lambda expression refactoring. Finally, an open response giving any additional comment to further explain their previous answers.

The results of the quantitative experiments were in contradiction. The Buse and Weimer model showed the comprehension to be unchanged when comparing code with and without lambda expressions, whereas the Posnett model showed there to be a decrease in comprehension. As well as these models, there was a test of whether the Cyclomatic Complexity was an accurate way to measure comprehension. Their analysis showed it to be a statistically insignificant metric to use when analysis, so this could affect the scoring of these models. Reduction in lines of code was shown to be a statistically sound metric for analyzing lambda expression comprehension.

The results of the qualitative experiments were spread across the two forms of snippet. For their first question, their findings showed that there was an indication of improved readability for developers. Then for the second question, they found that developers leaned more toward preferring the use of lambda expressions, but it was clear that the developers who had experience in functional programming were the ones who more-so preferred this. They then replicated the study for undergraduate students and found the results to be heterogeneous, that is that there was a positivity toward comprehension of lambda code between seasoned developers and newer developers.

3.2 A. Ward and D. Duego

The research of Ward [3] has set out to ask if the use of lambda expressions is faster than performing the same task with a non-lambda expression. To extract this data, they devised the following plan to achieve results:

- Source example uses of lambda expressions.
- Create their own lambda expression examples.
- Create the opposite non-lambda form to match each lambda-written example.
- Analyze the speed of the lambda expression task and compare with the speed of non-lambda

Their conducted research gained metrics from the comparison of reduction, filtering, collecting, mapping, and passing in a functions and predicates. Further their proposed dataset was comprised of 10,000 problems; the breakdown of lambda versus non-lambda was not given, so the assumed comparison is either 5,000-5,000 or 10,000-10,000. This would mean 5,000 lambdas and 5,000 non-lambdas. The resulting data can be shown in the table below:

Table 2: Lambda Performance Comparisons

Experiment	Lambda (ms)	Non-lambda (ms)	Lambda Improvement
Counting Primes	15.96	15.32	2.25
Adding Numbers	15.96	16.33	2.25
Concatenating Strings	30.44	72.82	58.20
Mapping	66.9	105.19	36.40
Filter List	69.21	105.54	34.42
Filter List with Predicate	74.71	106.63	29.93
Filter List in Function	81.15	107.71	24.66

Analyzing their data, it is clearly shown that the use of lambda expressions is more computationally efficient than the use of non-lambda expression. Especially in relation to concatenating strings, with an incredible 58.20ms difference.

An interesting item to note is that they did not source lambda expressions from live code, rather, they found code from textbook examples. This is where our study differs because we analyzed numerous open-source projects with live production code. We found this to be a more accurate depiction of the usage of lambda expressions.

4. Implementation

4.1 Research Questions

In order to guide our research into a significant direction, we formulated the following research questions:

- RQ1: Are Lambda Expressions used widely?
- RQ2: Does the use of Lambda Expressions reduce the lines of code?

4.2 Methodology

We selected 6 open-source projects to analyze for lambda expression use. We wanted to try to find a broad range of project sizes and functionality. As you can see in Table 3, our analyzed projects range from the smallest of 48,056 files in the Arduino project to the largest of 1,831,786 files in the Apache Flink project.

Table 3 Open-Source Projects

Project	Lambdas	LOC
Apache Solr	1,656	633,753
Arduino	337	48,056
Google Guava	168	517,803
Tomcat	60	446,444
Eclipse Smart Home	184	295,593
Apache Flink	4,828	1,831,786

To begin our analysis, we utilize GitHub to clone the projects. Once we have the projects in our directory, we then modify an existing framework that is used to analyze Java source code. To view a full analysis of the file history, we must break the program into what the file calls 'before' and 'after' files. This will be useful in determining whether a lambda expression is present before or after a commit to the repository.

One drawback to this framework is that it did not have the capability to pinpoint the use of lambda expressions. To resolve this, we had to create a formal Lambda Visitor that will be used to signify when a lambda expression is present in any given file. Once the file is visited, it must be sent to a python client from the Java Server in the following form:

$$Line\ Number : Container$$

Before we can parse the data returned from the Java Server, we must also insert the capability of breaking down the information in the above format. The information is then sent to a database for later analysis. In this database, we record the container granularity for each finding, the number of overall lambda expression occurrences for the specific project, and the file path for the given lambda expression.

5. Results

5.1 RQ1: Are Lambda Expressions widely used?

Our initial research question is whether lambda expressions are widely used in Java open-source projects. To test our question, we analyzed the breakdown of lambda expressions over the period of use. It should be noted that this feature was released to be downloaded in 2014, so all our data is subsequently after this date.

In our analysis scripts, we were able to use the revisions data from creating the 'before' and 'after' files to calculate the time in which the lambda expression was committed to the repository. Taking this data, we found it best to analyze via a line graph. This way we can view the entire trend from the start of the update to the current year of 2021.

Viewing the Figures 1 – 6, the trends are quite interesting. For Apache Solr, Apache Flink, and Google Smart Home projects the trend is almost continuously direct from the beginning of this feature until present day. As the projects gained in size, so did the number of lambda expressions in the code.

Although, this trend was not always the case. For the Arduino project, the trend was incredibly noisy. There was a large spike in lambda usage halfway through 2016 reaching above 250. Shortly after this, there was a sharp decrease by nearly 100 lambda expressions before the year 2017. Again, there was a large spike reaching its highest peak of 337 lambda expressions in 2019. Following the same trend as before, there was a sharp decrease of nearly 100 in lambdas in the same time frame. Our analysis shows a short life span of lambda adoption in this project. This could be due to deletion of various portion of the project, or simply factoring out the usage altogether. Further analysis of this would need to be done.

One item to note is that in every project we analyzed, there were high numbers of lambdas in the current year 2021. While some projects were slow to adopt the change, it does show a trend of heavy adoption eventually.

As well as visual data, we also display a table comparing the total lines of code, as well as the number of lambda expression in the project. The trend here is a harder to analyze. This could possibly be because there is no critical link between lines of code and the usage of lambda expressions, at least when simply viewing them side by side.

As you can see, the data varies heavily. The project entitled Apache Solr holds the next to most lambda expressions at 1656, while also having a substantial 633,753 lines of code. Again, you can see that the project entitled Apache Flink is the largest project in our research at 1,831,876 lines of code and subsequently 4,828 lambda expressions present.

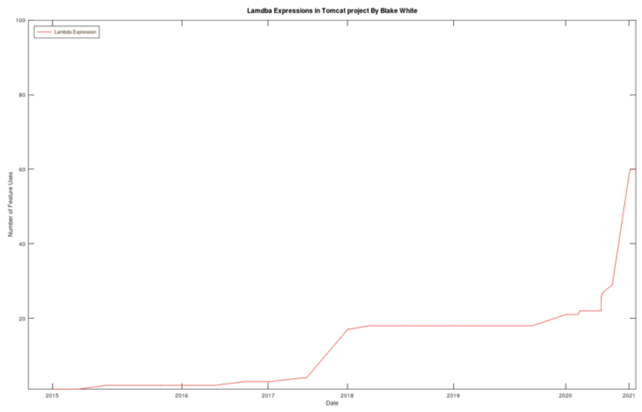

Figure 1: Tomcat

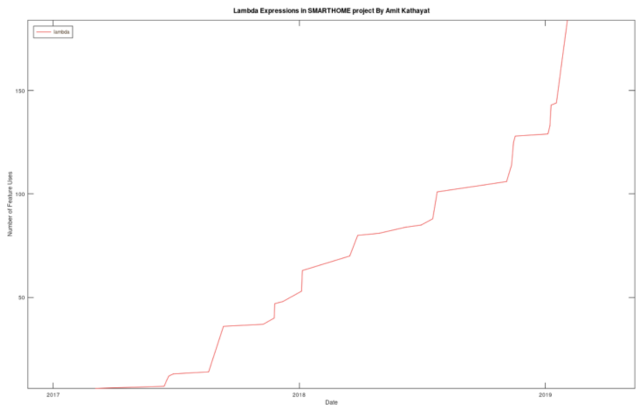

Figure 2: Smart Home

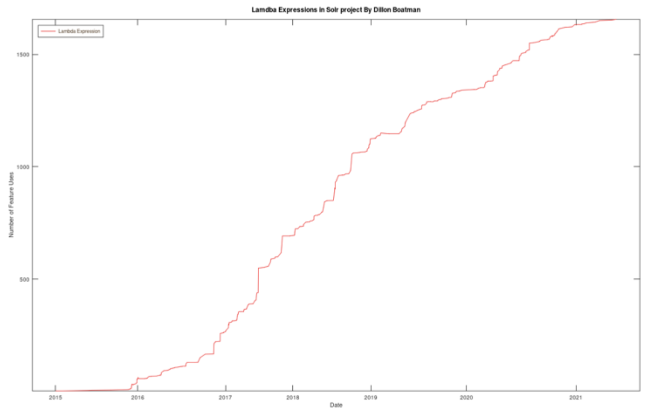

Figure 3: Apache Solr

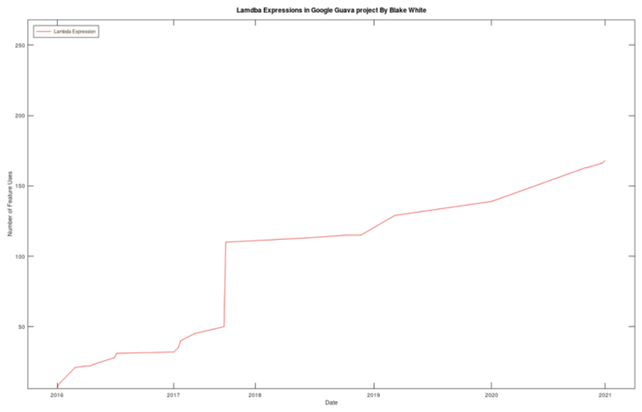

Figure 4: Google Guava

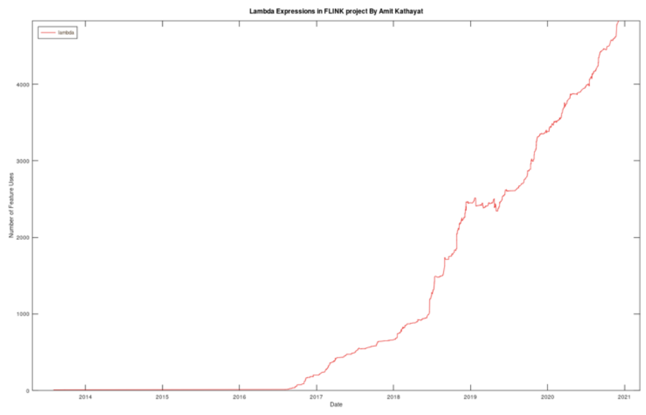

Figure 5: Flink

The correlation can be tried that with larger projects, there will be more lambda expressions; the rest of our data makes a counter argument. If you view the Google Guava project, with 517,803 lines of code, there are only 168 lambda expressions present. Even more peculiar, the smallest open-source project, Arduino, at 48,056 holds a considerable 337 lambda expressions. This would lean toward it being a domain, or even company specific choice to use lambda expressions over traditional code.

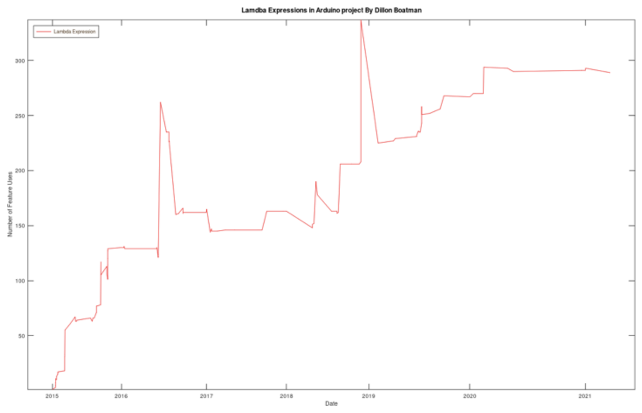

Figure 2: Arduino

5.2 RQ2: Does the use of Lambda Expressions reduce the lines of code?

Our second research question is whether lambda expressions reduce the lines of code in Java open-source projects. To test our question, we analyzed two lambda expressions from each open-source project.

In our analysis, we recorded the individual lines of code in a each chosen method. To record the change in lines, we converted the function to use generalized Java without lambda expressions. We decided to opt for the use of physical lines of code, rather than logical. With this being said, this could introduce bias if we are not careful. To negate any possible bias, we used the same style as the previous programmer, so as to not introduce any further lines of code by accident. An example of bias would be in the following:

```
for(int i = 0; i < 10; i++) // item 1
{
    System.out.println(i);
}

for(int i = 0; i < 10; i++){ // item2
    System.out.println(i);
}
```

Figure 8: Bias Representation

If you were wanting to introduce bias, you would convert the code to the loop from item 1 to item 2. This would resolve in a one line difference in either case. In our case, we preserved the previous style. This provides the most accurate indication that we could devise.

In our analysis, we found that in every open-source project that we analyzed methods of using a lambda expression, there showed a lines of code reduction. We do not make the claim that in every pre-conceivable instance of this expression there will be a code reduction, rather we simply did not notice anything aside from reductions.

Above you can see that the programmer is using a lambda function to return a 'blob' as a parameter. This is a perfect example of lambda usage, making it a one line of code difference.

```
public BlobContentRef<ByteBuffer> // 3
getBlobIncRef(String key) {
    return getBlobIncRef(key, () ->
addBlob(key));
  }

public BlobContentRef<ByteBuffer> // 4
getBlobIncRef(String key) {
    BlobContent<ByteBuffer> blob =
addBlob(key);
    return getBlobIncRef(key, blob);
  }
```

Figure 8: Apache Solr Conversion

As another example below, you can see that the lambda was essentially being used in the same manner as before in the parameter of a function. Again, this can be done without a lambda expression but it would add an additional line of code.

```
private String
createOperationKey(OperationInfo operation)
{
        StringBuilder key = new
StringBuilder(operation.getName());
        key.append('(');

StringUtils.join(operation.getSignature(),
',', (x) -> x.getType(), key);
        key.append(')');
        return key.toString();
}

private String
createOperationKey(OperationInfo operation)
{
        StringBuilder key = new
StringBuilder(operation.getName());
    String xType = x.getType();
        key.append('(');

StringUtils.join(operation.getSignature(),
',', xType, key);
        key.append(')');
        return key.toString();
}
```

Figure 9: Tomcat Conversion

Our results from this analysis showed an average of 1.7 lines of code reduction over the 12 extracted and converted lambdas. This is not an intense amount, but it seems to be a consistent reduction.

6. CONCLUSION

In this project we analyzed 6 open-source projects using an existing framework that we adapted to find and extract the information from all lambda uses in each file. Throughout this analyzing, we found that the use of lambda expressions showed to be popular amongst all projects. While the adoption rate was slow for some, there was a consistent increase in the adoption rate near the year 2021 for all projects analyzed. As well as this, we found that the use of lambda expressions does provide an average of 1.7 reduction in lines of code

7. REFERENCES

[1] Walter Lucas, Rodrigo Bonifácio, Edna Dias Canedo, Diego Marcílio, and Fernanda Lima. 2019. Does the Introduction of Lambda Expressions Improve the Comprehension of Java Programs? In Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES 2019). Association for Computing Machinery, New York, NY, USA, 187–196. Ding, W. and Marchionini, G.

1997. *A Study on Video Browsing Strategies*. Technical Report. University of Maryland at College Park.

[2] Lambda Expression in Java https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/Lambda-QuickStart/index.html

[3] A. Ward, D. Deugo. 2015. Performance of lambda expressions in java 8. Athens: The steering Committee of the World Congress in Computer Science, Computer Engineering and Applied Computing.

[4] Donghoon Kim, Gangman Yi. Measuring Syntactic Sugar Usage in Programing Languages: An Empirical Study of C# and Java Projects.