HIGH SCHOOL EDITION
Journal of Student Research
Volume 12 Issue 1 (2023)

Insights to Improve Computer Science Education by Enhancing the Java Language and Tools

Pranav Eluri[1] and Domingo David[1#]

[1]Westview High School, San Diego, CA, USA
[#]Advisor

ABSTRACT

Java is the most well-known and widely used object-oriented programming language. It uses "objects" as instances of particular data types, or classes. While many computer science students who take Java programming beginner courses, such as AP Computer Science A and fundamental college Java coding courses, succeed in learning Java fundamentals, some aspects of Java make it less intuitive than other programming languages and more difficult to understand than necessary. The purpose of this study is to present eleven insights on how Java could be made more learner-friendly based on the author's personal experience learning Java and analyze whether these eleven insights are consistently perceived as useful to current student learners and industry practitioners. A digital survey was conducted with individuals who are learning or have learned the language. Respondents were presented with questions and were instructed to choose the best answer to each question. The results from the survey indicated that four insights were perceived as useful and that there was no inherent preference for the remaining six insights. Further research must be conducted to fully understand the perception differences between new learners and experienced practitioners.

Literature Review

Ogbuokiri et al. (2016) conducted an analysis of Python and Java to determine which programming language is best for a beginner. The study compared the programming complexity of using common syntax features in both languages and also compared the number of lines of code (LOC), file capacity, and speed by implementing QuickSort and Tic-Tac-Toe games in both Java and Python. They concluded that both languages have advantages and disadvantages, and it is impossible to say with certainty that one is superior to the other. They found that although the structure of Java is slightly more complex than that of Python, it provides a better understanding of memory management and is more secure. On the other hand, they found Python to be short, simple, and straightforward; easy to understand because it has simple English-like syntax; and easy to read because indentation is required in Python. In Java, the use of a semicolon to indicate the end of a line is sometimes overlooked, resulting in a major compilation error. Because Python is dynamically typed, the variable type is checked during run time, whereas Java is statically typed, so the exact datatype for variables is known during compilation, resulting in faster execution than Python. Because most programming languages have similar fundamentals, regardless of which language is chosen, a person can easily learn another programming language. Ogbuokiri et al. (2016) recommend that the novice not keep changing the language until they have mastered it, as it would lead to a loss of confidence.

Bloch (2000) compares his first experiences teaching the programming languages Java and Scheme and concludes that Java emphasized syntax and platform, whereas Scheme emphasized data structures, function composition, and software engineering.

Bloch (2000) made the following observations regarding Java instructions:

1. It took longer for students and instructors to comprehend syntax and become comfortable with the development process, leaving little time for program design and software engineering principles.
2. The majority of students felt that Java programming was tedious, frustrating, and required memorization of several rules that did not always make sense to first-time programmers, as well as a tedious and frustrating clean compile. Creating a functional program was not simple.
3. While object-oriented programming was not difficult for novice programmers, Java felt difficult, maybe because it has so many syntax rules that cause programs to be much longer than necessary.
4. Almost all Java development platforms were created with experienced programmers in mind, not novices. Error messages were ambiguous, requiring the user to set multiple options even when creating the simplest programs, which allowed beginners to inadvertently invoke advanced language features. For instance, a misplaced curly brace in Java produces an "inner class" that generates a series of misleading error messages.
5. The majority of students have difficulty analyzing errors. Their approach to debugging is random, and they do not know how to break down errors when a program fails to function.
6. Students frequently spend a considerable amount of time correcting errors. The majority of students, especially college freshmen, must start projects early to meet deadlines.
7. Students frequently confuse accidental platform and language syntax issues with fundamental software program design and software engineering issues.

Jegede (2019) examines various error types and patterns in the Java programming language based on the fundamental concepts of methods, classes, conditional structures, looping, and other fundamental concepts of objects for novice programmers of varying skill levels. The most common type of error is missing symbols, while invalid symbols are the least common. Advanced concepts such as methods and classes had the highest number of errors, while fundamental constructs such as loops had the lowest. The same error types occurred across all ability levels. As anticipated, low-achieving students had greater difficulty writing bug-free code for more complex concepts. This demonstrates that more advanced Java concepts, such as methods and classes, are difficult to program, whereas conditional and looping constructs, which are common to many programming languages, are simple for all programmers to program. This suggests that it is difficult to master the syntax of more advanced Java concepts, and a number of potential enhancements are possible. Beginner programmers frequently find it difficult to write syntactically correct code (Jadud 2005, Jadud 2006, and Nienaltowski et al., 2008). Furthermore, programming courses that emphasize algorithms and problem-solving frequently expect students to deal with syntax issues when completing programming assignments and lab exercises (Edwards et al. 2008, Parlante 2008, and Stevenson et al. 2006). To determine the extent to which syntax is a barrier in a Java-based computer science course, Denny et al. (2011) analyzed code from a few hundred students for the correctness of the code submitted as part of a practice drill consisting of short Java programming exercises. They divided students into four quartiles, with the top performers in the first quartile. Students in all quartiles were writing code with syntax errors that were difficult to compile, and weaker students were unable to solve syntax problems in the majority of instances. Syntax errors rather than logic errors were the most common cause of failures. The number of syntax errors in the code written by students in the top quartile was also very high, possibly because these students were attempting to solve more difficult problems and encountered a wider variety of syntax errors than weaker students. Denny et al. (2011) conclude that Java's syntax is difficult to use and can be a barrier to learning the language, but that it can be made more user-friendly through a number of modifications.

The Problem

As summarized in the above literature review, Java is difficult to learn. Based on the author's personal experience as a beginner learning Java, we share eleven insights to improve Java language learning and increase adoption by beginners learning Java. The insights are divided into five themes, which are discussed further below.

Theme 1: Make Java a true Object-Oriented Programming Language

Java (Arnold et al. 2000) is designed to make it easier for C and C++ programmers to migrate to Java. Java, for example, has a few procedural features like primitive types and primitive operators. Java's procedural features prevent it from becoming a purely object-oriented language. Eliminating procedural features eliminates the need for beginners to learn procedural concepts, allowing them to focus their energy on only object-oriented concepts, as described next.

Insight 1: Eliminate Primitive Types

Java has eight primitive types, which are predefined by the language and named by a reserved keyword. The eight primitive data types supported by the Java programming language are shown in Table 1.

Table 1: Primitive Data Types (JDK 19 documentation 2022)

S.No	Data Type	Length
1	byte	8-bit
2	short	16-bit
3	int	32-bit
4	long	64-bit
5	float	32-bit IEEE 754 floating point values
6	double	double-precision 64-bit IEEE 754 floating point values
7	boolean	only two possible values: true and false
8	char	single 16-bit Unicode character

Java provides object types, which are based on object-oriented concepts, in addition to primitive types. Java includes many predefined object types as well as the ability to create user-defined object types. As shown in Listing 1, replacing all primitive types with corresponding predefined object types eliminates the need for procedural primitive constructs, allowing beginners to focus on learning object-oriented constructs and transforming Java into a pure object-oriented language. Some programs may get more complicated (Biddle et al., 1998), but the authors think that's because objects and object-oriented ideas aren't taught well enough at the start of a Java course.

Listing 1: Replacing primitive types with Object types.

```
Class Example1 {
        public static void  main(String args[])  {
        /*
        byte b = 1;
        short s = 2;
        int i = 3;
        long l= 4L;
        float  f=5.0F;
        double  d = 6.0D;
        Boolean bool= true;
        char c ='a';
         */
        // The above code could be replaced by the following code.
        Byte b = 1;
        Short s =2;
        Integer  i = 3;
        Long  l= 4L;
        Float  f = 5.0F;
        Double d = 6.0D;
        Boolean bool = true;
        Character c = 'a';
  }
}
```

Insight 2: Eliminate Primitive Operators

Java has predefined primitive operators as well as the ability for programmers to create user-defined operators (also known as methods). The language defines primitive operators by using special symbols that perform specific operations on one, two, or three operands and then return a result. These operators have difficult-to-remember precedence, left association, and right association rules. All of the primitive symbol-based operators are listed in Table 2 in precedence order. The higher an operator's precedence, the closer it appears to the top of the table. Operators with higher precedence are evaluated first, followed by operators with lower precedence. Operators on the same line have equal precedence. When two operators with equal precedence appear in the same expression, a rule must be applied to determine which is evaluated first. Except for the assignment operators, all binary operators are evaluated from left to right; assignment operators are evaluated from right to left.

Eliminating operator symbols and replacing each of the operators with a descriptive method would eliminate the need for beginners to learn precedence rules, resulting in faster learning, less confusion (Biddle et al. 1998), the elimination of procedural primitive constructs, and the transformation of Java into a pure object-oriented language. An example of replacing operators with methods is shown in Listing 2. While method names may make some programs more complicated, the authors believe that the ease of program operator precedence readability will offset this complication.

Listing 2: Replacing primitive operators with methods.

```
class Example2 {
   public static void main(String args[]){
        int i1 = 1;
        int i2 = 2;
        i2 = i2 + i1;
        i2 = i2 * i1;
        i2 = i2/i1;

        // Replace the primitive type operations with methods as shown below
        Integer I1 = 1;
        Integer I2 = 2;
        I2.add(I1);
        I2.multiply(I1);
        I2.divide(I1);
  }
}
```

Theme 2: Simplify Types

While Java could be made a pure object-oriented language by removing primitive types and primitive operators, this may not be possible due to its widespread use. In this case, simplifying data types by reducing the number of primitive and non-primitive data types and replacing similar operators eliminates the need for beginners to learn multiple data type concepts, removes confusing operators, and simplifies Java.

Table 2: Primitive Operators (JDK 19 documentation, 2022)

S.No	Operators	Precedence
1	Postfix	expr++ expr–
2	Unary	++expr –expr +expr -expr ~!
3	multiplicative	* / %
4	Additive	+ -
5	Shift	& << >> >>>
6	relational	< > ≤ ≥ instanceof
7	Equality	== !=
8	bitwise AND	&
9	bitwise exclusive OR	^
10	bitwise inclusive OR	|
11	logical AND	&&
12	logical OR	||
13	ternary	? :
14	assignment	= += -= *= /= %= &=^= |= <<= >>= >>>=

Insight 3: Replace ten numeric types with one number type

Java has ten numeric types, five of which are primitive types (shown in Table 1), and five corresponding object types. Replacing these ten numeric types with a single number type will simplify the language, make programs readable, aid in remembering the concept clearly, and avoid type confusion. Based on the context, the number type can abstract the various types and store the appropriate specific numeric type and required precision. Listing 3 shows an example of replacing ten numeric types with one number type.

Listing 3: Replacing ten numeric types with one number type.

```
Class Example3 {
        Public static void main(String args [ ]){
           byte  b1 = 1;
           short  s1 = 2;
           int  i1 = 3;
           long  l1 = 4L;
           float f1 =  5.0F;
           double d1 =  6.0D;

           Byte  b2 = 1;
           Short  s2 = 2;
           Integer i2  = 3;
           Long l2  = 4L;
           Float f2 = 5.0F;
           Double  d2 = 6.0D;

           //all of the above statements could be replaced with
           Number  b1 = 1;
           Number s1  = 2;
           Number i1  = 3;
           Number l1  = 4L;
           Number at f1 =  5.0F;
           Number d1 =  6.0D;

           Number b2  =  1;
           Number s2  =  2;
           Number i2  = 3;
           Number l2  =  4L;
           Number f2  =  5.0F;
           Number d2  =  6.0D;
        }
}
```

Insight 4: Replace similar operator symbols with a method

The operator symbols =, ==, and .equals stand for assignment, primitive type and object reference equality checking, and object equality checking, respectively. This operator usage causes confusion (Biddle et al. 1998), which can be

avoided by replacing each operator with a method that describes the operations. For example, = should be replaced by the "set" method, == by the "isEqual" method, and "equals" by the "isEqual" method.

Theme 3: Simplify Coding

Java provides many ways of accomplishing the same task. There are four ways to write loops, three ways to write condition statements, and three ways to increment a variable. While it is ideal to provide only one way to complete a task, it is preferable to avoid adding too many ways to complete the same task and adding complexity to the language. Reducing the number of ways code can be written to accomplish a particular task reduces the need for beginners to learn multiple code constructs, increases code readability, and simplifies Java learning.

Insight 5: Replace four iteration constructs with one construct

Java provides four ways for writing iterations: for loop, enhanced for loop, while loop, and do-while loop. While there are times when each iteration is appropriate, it does complicate learning a new language. To simplify code, using a single loop for all iteration use cases could be a reasonable compromise. Listing 4 shows an example of each of the four ways to write iteration statements.

Listing 4: Four ways to write iteration statements.

```
class Example4 {
        public static void main(String args []){
        // simple for loop
        for(int i = 0; i<10; i++) {
            //do something
        }

        //enhanced for loop
        int [] range = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
        for(int i: range) {
            //do something
        }

        //while loop
        int i = 0;
        while(i <10) {
            //do something
           i++;
        }
        //do-while loop
        int j = 0;
        do {
            //do something
           j++;
        } while (j > 9);
  }
}
```

Insight 6: Replace three conditional constructs with one construct

There are three ways to write a conditional statement in Java: if, switch, and ternary operators. Furthermore, the "if" statement has several extensions. While there are times when each conditional statement is appropriate, it adds an unnecessary burden to the new language learner. To simplify code and improve learning, using a simple "if" statement for all conditional use cases would be a good compromise. Listing 5 shows an example of each of the three ways to write a conditional statement.

Listing 5 Three ways to write a conditional statement.

```
class Example5 {
   public static void main(String args []){
    int oneCount = 0;
    int twoCount = 0;
    int otherCount = 0;
    int colorCode = 1;

    //using if statement

    if(colorCode == 1) {
      oneCount++;
    } else if (colorCode == 2) {
      twoCount++;
    } else {
      otherCount++;
    }

   // using switch statement
   switch (colorCode) {
      case 1:
        oneCount++;
        break;

      case 2:
       twoCount++;
       break;

       default:
        otherCount++;
   }

   // using ternary operator
   int x = color Code == 1 ? oneCount++ :
            color Code == 2 ? twoCount++ : otherCount++;
  }
}
```

Insight 7: Replace three increment statements with one

Java provides three ways for increasing a variable: i = i + 1, i ++, and i+ = 1. While there are times when each increment statement is appropriate, it adds unnecessary confusion because it differs from the standard mathematical way of adding values to a variable. Using mathematical standards with which the student is familiar to complete tasks similar to mathematics would alleviate any anxiety associated with learning to code.

Theme 4: Java Tools

Eclipse, IntelliJ IDEA, and NetBeans are among the most popular Java integrated development environments (IDEs). While each of these environments provides a variety of tools in addition to the ones included with the Java installation, some core tool enhancements would make Java more powerful.

Insight 8: Tool to add parenthesis

While some Java IDEs include formatting features and the ability to insert parenthesis when writing new code, none of them can automatically add parentheses for order of operation unless the coder adds their own. When debugging, a command-line tool for adding parentheses would save novice programmers a considerable amount of time.

Insight 9: Tool to auto-correct mistakes

Java, like other modern technologies such as English language editors, should provide a tool to “autocorrect” code as needed to correct mistakes; at the very least, it should have a “spell-check” type feature. These could correct spelling errors in keywords, mistyped variables based on context, and mistyped index variables when the usage is clearly incorrect.

Insight 10: Tool to auto-fill import statements

While some Java IDEs include auto-filling import features, none of them can automatically add import statements. When debugging, a command-line tool for adding import statements would save novice programmers a considerable amount of time.

Theme 5: Teaching Java

Prior research (Jegede et al. 2019) discovered that class writing, method writing, and syntax errors were extremely common. According to Topolnik (2005), the syntax for Java can be more difficult than for other languages. Programming classes begin with procedural concepts, making learning object-oriented concepts more difficult when they are introduced as an additional concept midway through the semester. Instead, students would learn and appreciate object-oriented programming if they began with object-oriented concepts as their fundamental way of thinking.

Insight 11: First, focus on object-oriented concepts

Instead of teaching procedural concepts, high schools should begin with object-oriented programming. Ideally, students should learn the definitions of an object and a class, as well as the concept of inheritance, including sub-classes and super-classes. Variable coding, conditional statements, iterations, and methods should be explained in relation to

object-oriented concepts. The coding aspect would then be better comprehended in the context of object-oriented programming.

Methods

Summary of Method

The purpose of this study is to present eleven insights on how Java could be made more user-friendly based on the author's personal experience learning Java and analyze whether these eleven insights are consistently perceived as useful to current student learners and industry practitioners. A digital survey was conducted with individuals who are learning or have learned the language. Respondents were presented with three types of questions: demographic questions, Java knowledge testing questions, and insight validation questions. Demographic questions are shown in Table 3 and were used to get residence, grade, gender, and experience information. Java knowledge testing questions are shown in Table 4 and were used to test Java knowledge. Insight validation questions are shown in Table 5, and respondents were instructed to choose an agree or disagree answer to each question. The results of these were analyzed with the chi-square goodness of fit test. The chi-square goodness of fit test determines whether a sample's proportions of discrete outcomes follow a population distribution with hypothesized proportions. In other words, do the observed proportions match the values suggested by theory when a random sample is drawn? The chi-square goodness of fit test determines whether the proportions of discrete outcomes are all equal, though the proportions used in the test can be specified.

Table 3: Demographic Survey Questions.

Q. No	Question
1	In what country do you live?
2	What grade are you in?
3	What is your gender?
4	How many years of Java experience do you have?

Table 4: Java Knowledge Testing Survey Questions.

Q. No	Question
1	In how many ways can your write iterations in Java?
2	In how many ways can you write conditional statements in Java?
3	How many ways can your write increment statements in Java?

Participants

As the survey was related to Java programming, it was sent to individuals who are learning Java or are using Java in their work environment. To solidify this assumption, the survey asked the participants to share the number of years of experience they have with Java. The survey was sent to 100 individuals, and out of these recipients, we received 50 responses, providing an overall response rate of 50%.

Table 5: Survey Questions.

Q. No	Question
1	Replacing primitive types (byte, short, int, long, float, double, boolean, and char) with object types (Byte, Short, Integer, Float, Double, Boolean, and Char) would make learning Java easier.
2	Replacing primitive operators (+, -, *, /, ^, etc.,) with method names like (add, subtract, multiply, divide, exponentiation) would help avoid having to memorize precedence rules.
3	Replacing ten numeric types in Java with a single number type would simplify the Java language and allow me to learn it more quickly.
4	The operator symbols =, ==, and .equals, respectively, represent assignment, primitive type and object reference equality checking, and object equality checking. Replacing each operator with a method that describes the operation would be less confusing.
5	Using a single loop construct type ("for loop") for all iteration types would be easier to understand and would help reduce confusion.
6	Having just one kind of conditional statement ("if-statement") would make Java less confusing than having multiple kinds of conditional statements.
7	Java would be less confusing if it had only one kind of increment operator (example: i = i + 1).
8	A tool that adds parentheses would save debugging time.
9	A tool that “autocorrect” code as needed to correct syntax errors would save debugging time.
10	A tool that that auto-fills import statements would reduce the time it takes to search for packages in the Java libraries.
11	Instead of first teaching procedural concepts, beginning with object-oriented programming, would help novice Java learners better understand object-oriented programming.

Design

For each of the eleven questions listed in Table 5, participants were asked to choose one of five responses, namely, “strongly agree,” “agree,” “neutral,” “disagree,” and “strongly disagree.” All questions are directly related to the insights. There were additional questions on Java experience, education, country of residence, and gender, which are used in the discussion section of the study. For analyzing the results, Strongly Agree and Agree counts are treated as Agree count, Strongly Disagree and Disagree are treated as Disagree. Neutral counts are not used. With the process of counting, we had 36 responses.

Results

The survey received 50 responses. Of these responses, 14 used the neutral position and were discarded. Of the 36 respondents, 22 are men and 14 are women. 24 respondents live in the US, and the remaining 12 live in India. Of the 36 respondents, 1 is in grade 10, 7 are in grade 11, 7 are in grade 12, and 2 have completed grade 12. All 36 respondents have Java programming experience: 1 with 1 year of experience, 2 with 2 years of experience, 15 with 3 years of experience, 5 with 4 years of experience, 2 with 5 years of experience, and 10 with more than 5 years of experience.

Table 6 depicts the preferences for the eleven insights. The columns show the question number, total response count, agreement response count, disagreement response count, and Chi-square Goodness of Fit result. The result shows that only four out of eleven are statistically significant.

Table 6: Survey Responses.

Q. No	Total Responses	Agree	Disagree	Chi Square Goodness of Fit Result
1	36	18	18	The Chi^2 value is 0. The *p*-value is 1. The result is *not* significant at $p < .05$.
2	36	17	19	The Chi^2 value is 0.111. The *p*-value is .73888. The result is *not* significant at $p < .05$
3	36	21	15	The Chi^2 value is 1. The *p*-value is .31731. The result is *not* significant at $p < .05$.
4	36	23	13	The Chi^2 value is 2.778. The *p*-value is .09558. The result is *not* significant at $p < .05$.
5	36	20	16	The Chi^2 value is 0.444. The *p*-value is .50499. The result is *not* significant at $p < .05$.
6	36	20	16	The Chi^2 value is 0.444. The *p*-value is .50499. The result is *not* significant at $p < .05$.
7	36	21	15	The Chi^2 value is 1. The *p*-value is .31731. The result is *not* significant at $p < .05$
8	36	33	3	**The Chi^2 value is 25. The *p*-value is < .00001. The result is significant at $p < .05$.**
9	36	29	7	**The Chi^2 value is 13.444. The *p*-value is .00025. The result is significant at $p < .05$**
10	36	29	7	**The Chi^2 value is 13.444. The *p*-value is .00025. The result is significant at $p < .05$.**
11	36	26	10	**The Chi^2 value is 7.111. The *p*-value is .00766. The result is significant at $p < .05$.**

Discussion

As summarized in Table 6, only four out of the eleven tested insights have statistically significant results. Below, we discuss the results further with additional Java knowledge questions administered in the survey.

Theme 1: Make Java a true Object-Oriented Programming Language

Java's procedural features, like primitive types and primitive operators, prevent it from becoming a purely object-oriented language. Eliminating procedural features eliminates the need for beginners to learn procedural concepts, allowing them to focus their energy on only object-oriented concepts. Based on the survey, there is merit to this insight, but we could not find a statistically significant difference supporting this insight. While 16 out of the 36 agreed that "primitive operator precedence rules were confusing," 20 disagreed. So we conclude that this is a personal preference.

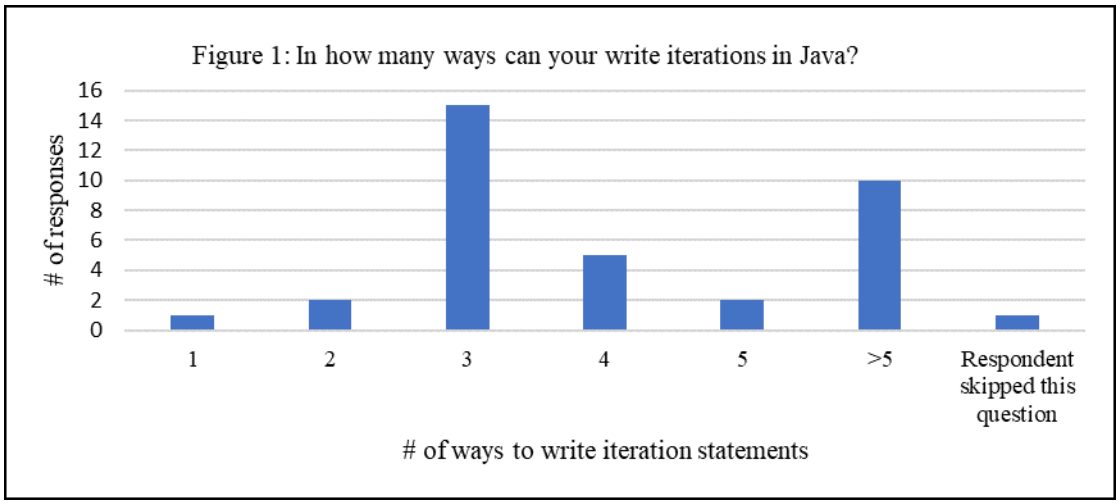
Theme 2: Simplify Types

Twenty-three out of 36 respondents agree that simplifying data types by reducing the number of primitive and non-primitive data types eliminates the need to learn multiple data type concepts, but the result is not statistically significant. There are two ways to further test this hypothesis: one is with a larger sample, and the second is A/B testing, with one group (A) taught with only one data type and the other group (B) taught with standard Java types, and seeing how they perform in an exam.

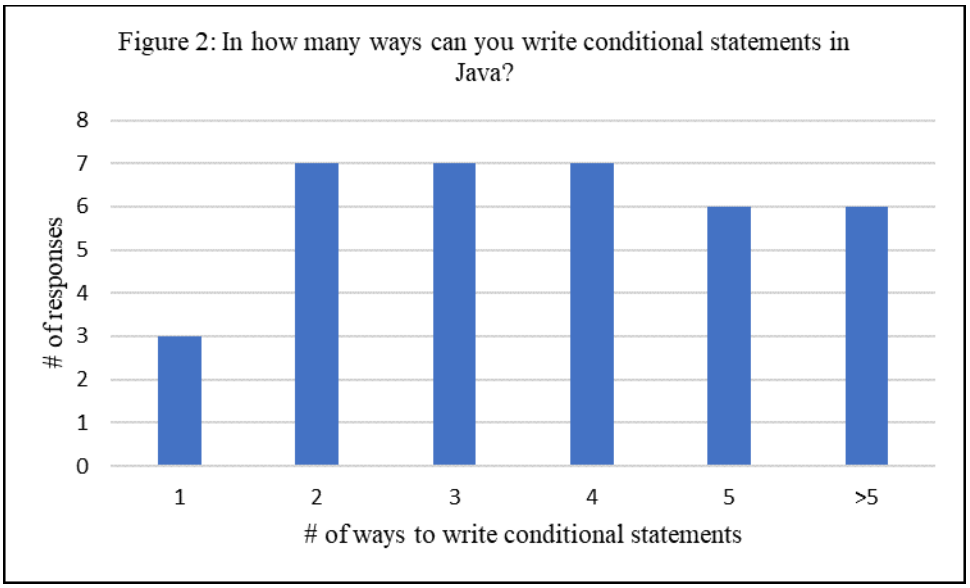
Theme 3: Simplify Coding

Java provides many ways of accomplishing the same task. There are four ways to write loops, three ways to write condition statements, and three ways to increment a variable. While it is ideal to provide only one way to complete a task, it is preferable to avoid adding too many ways to complete the same task and adding complexity to the language. Reducing the number of ways code can be written to accomplish a particular task reduces the need for beginners to learn multiple code constructs, increases code readability, and simplifies Java learning.

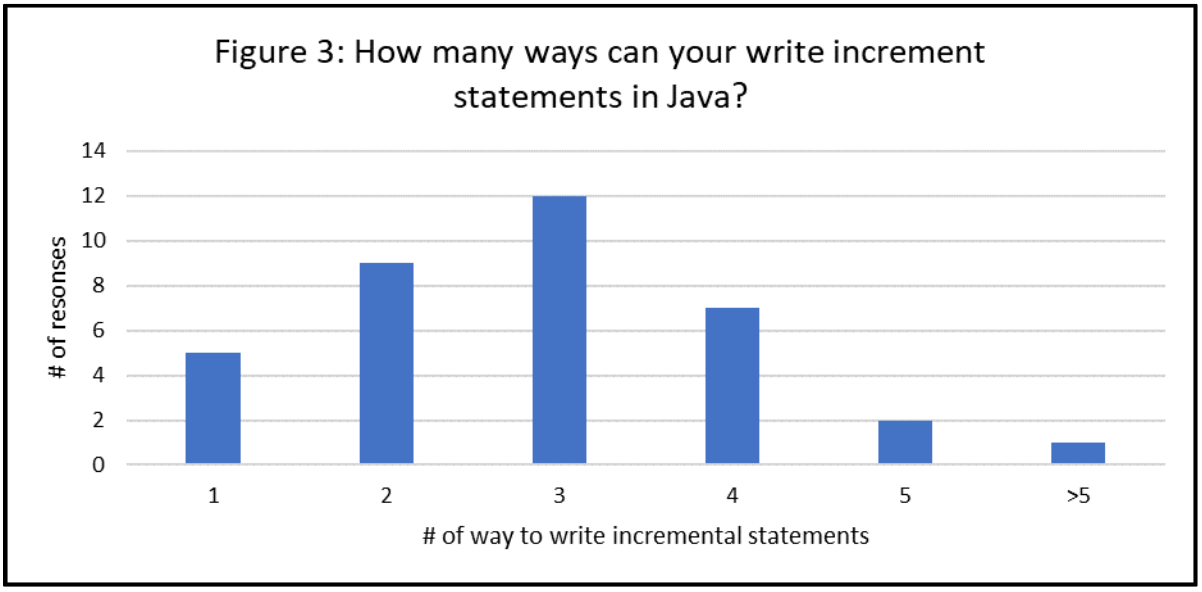
There are four ways to write iteration statements, but as shown in Figure 1, most respondents thought there were only three ways to do so. Even though most respondents did not remember the number of ways accurately, only 20 out of 36 were in favor of limiting the number of iteration statements to 1. While there is support for the idea, it is not statistically significant.



There are three ways to write conditional statements, but as shown in Figure 2, respondents chose all options almost equally. Even though respondents did not remember the number of ways accurately, only 20 out of 36 were in favor of limiting the number of conditional statements to 1. While there is support for the idea, it is not statistically significant.



There are three ways to write increment statements, but as shown in Figure 3, most respondents chose three, with two and four as the next highest options. Even though respondents did not remember the number of ways accurately, only 21 out of 36 were in favor of limiting the number of increment statements to 1. While there is support for the idea, it is not statistically significant.



Theme 4: Java Tools

Eclipse, IntelliJ IDEA, and NetBeans are among the most popular Java integrated development environments (IDEs). While each of these environments provides a variety of tools in addition to the ones included with the Java installation, some core tools are missing. As hypothesized, there was statistically significant support for providing tools for adding parenthesis, auto-correcting mistakes, and automatically filling import statements.

Theme 5: Teaching Java

The idea of teaching object-oriented programming rather than procedural programming was supported by statistically significant results. Twenty-six out of 36 respondents favored this approach.

Conclusion

Java is a highly useful and popular programming language, and its popularity among novice programmers is growing. Because more people are learning to use Java, it is worthwhile to make it more user-friendly. In this paper, we discuss how to make it easier for novices to learn Java by enhancing the language, its tools, and its teaching methods. In the future, to further validate the efficacy of our proposed insights, we intend to conduct a survey of the learning challenges faced by new Java learners and collect additional data regarding Java learning.

Limitations

One limitation is the generalizability of the conclusions. Given that we have a survey with only 36 respondents, caution is needed to generalize the conclusion. A survey with a larger sample covering a wider demographic population in terms of grade, country of residence, and Java experience will help generalize the results. Though there are limitations with the current survey, it is still a good mechanism to establish that Java language enhancements are directionally needed.

Another limitation of the method is the type of study. While surveys establish conclusions directionally, they have various biases, like leading questions, selection biases, and response biases. To confidently establish the conclusion, we would need a controlled experiment with students who are learning the language for the first time.

References

Jegede, Philip & Olajubu, Emmanuel & Ejidokun, Adekunle & Elesemoyo, Isaac. (2019). Concept–based Analysis of Java Programming Errors among Low, Average and High Achieving Novice Programmers. Journal of Information Technology Education: Innovations in Practice. 18. 49-59. https://doi.org/10.28945/4322

Bloch, Stephen. (2000). Scheme and Java in the first year. Journal of Computing Sciences in Colleges. 15. 157-165.

Ogbuokiri, Blessing & Agu, Monica & B.O, Okwume. (2016). Comparison of python and java for use in instruction in first course in computer programming. Transylvanian Review. 24.

JDK 19 documentation - home. Oracle Help Center. (2022, September 21). Retrieved November 11, 2022, from https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Biddle, R., & Tempero, E. (1998). Java pitfalls for Beginners. ACM SIGCSE Bulletin, 30(2), 48–52. https://doi.org/10.1145/292422.292441

Arnold, K., & Gosling, J. (2000). The Java programming language. Addison-Wesley.

Edwards, S. H., Börstler, J., Cassel, L. N., Hall, M. S., & Hollingsworth, J. (2008). Developing a common format for sharing programming assignments. ACM SIGCSE Bulletin, 40(4), 167–182. https://doi.org/10.1145/1473195.1473240

Jadud, M. C. (2005). A first look at novice compilation behaviour using bluej. Computer Science Education, 15(1), 25–40. https://doi.org/10.1080/08993400500056530

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. Proceedings of the 2006 International Workshop on Computing Education Research - ICER '06. https://doi.org/10.1145/1151588.1151600

Nienaltowski, M.-H., Pedroni, M., & Meyer, B. (2008). Compiler error messages. ACM SIGCSE Bulletin, 40(1), 168–172. https://doi.org/10.1145/1352322.1352192

Parlante, N. (2008). Nifty assignments. ACM SIGCSE Bulletin, 40(1), 112–113. https://doi.org/10.1145/1352322.1352173

Stevenson, D. E., & Wagner, P. J. (2006). Developing real-world programming assignments for CS1. ACM SIGCSE Bulletin, 38(3), 158–162. https://doi.org/10.1145/1140123.1140167

Denny, P., Luxton-Reilly, A., Tempero, E., & Hendrickx, J. (2011). Understanding the syntax barrier for novices. Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education - ITiCSE '11. https://doi.org/10.1145/1999747.1999807

Topolnik, M. (2005). An improved XML syntax for the java programming language. Proceedings of the 8th International Conference on Telecommunications, 2005. ConTEL 2005., 2, 485-492.