Journal of Information Technology Education: Innovations in Practice

*An Official Publication of the Informing Science Institute*
*InformingScience.org*

*JITEiip.org*

**Volume 18, 2019**

# CONCEPT–BASED ANALYSIS OF JAVA PROGRAMMING ERRORS AMONG LOW, AVERAGE AND HIGH ACHIEVING NOVICE PROGRAMMERS

| | | |
|---|---|---|
| Philip Olu Jegede* | Institute of Education, Obafemi Awolowo University, Ile-Ife, Nigeria | pojegede@gmail.com |
| Emmanuel Ajayi Olajubu | Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria | emmolajubu@oauife.edu.ng |
| Adekunle Olugbenga Ejidokun | Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria | gbengskul@gmail.com |
| Isaac Oluwafemi Elesemoyo | Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria | ielesemoyo@gmail.com |

*Corresponding author

## ABSTRACT

| | |
|---|---|
| Aim/Purpose | The study examined types of errors made by novice programmers in different Java concepts with students of different ability levels in programming as well as the perceived causes of such errors. |
| Background | To improve code writing and debugging skills, efforts have been made to taxonomize programming errors and their causes. However, most of the studies employed omnibus approaches, i.e. without consideration of different programing concepts and ability levels of the trainee programmers. Such concepts and ability specific errors identification and classifications are needed to advance appropriate intervention strategy. |
| Methodology | A sequential exploratory mixed method design was adopted. The sample was an intact class of 124 Computer Science and Engineering undergraduate students grouped into three achievement levels based on first semester perfor- |

Accepting Editor Anthony Scime | Received: January 7, 2019 | Revised: March 23, April 12, April 21, 2019 | Accepted: April 24, 2019.

Cite as: Jegede, P. O., Olajubu, E. A., Ejidokun, A. O., & Elesemoyo, I. O. (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

| | |
|---|---|
| | mance in a Java programming course. The submitted codes in the course of second semester exercises were analyzed for possible errors, categorized and grouped across achievement level. The resulting data were analyzed using descriptive statistics as well as Pearson product correlation coefficient. Qualitative analyses through interviews and focused group discussion (FGD) were also employed to identify reasons for the committed errors. |
| Contribution | The study provides a useful concept-based and achievement level specific error log for the teaching of Java programming for beginners. |
| Findings | The results identified 598 errors with Missing symbols (33%) and Invalid symbols (12%) constituting the highest and least committed errors respectively. Method and Classes concept houses the highest number of errors (36%) followed by Other Object Concepts (34%), Decision Making (29%), and Looping (10%). Similar error types were found across ability levels. A significant relationship was found between missing symbols and each of Invalid symbols and Inappropriate Naming. Errors made in Methods and Classes were also found to significantly predict that of Other Object concepts. |
| Recommendations for Practitioners | To promote better classroom practice in the teaching of Java programming, findings for the study suggests instructions to students should be based on achievement level. In addition to this, learning Java programming should be done with an unintelligent editor. |
| Recommendations for Researchers | Research could examine logic or semantic errors among novice programmers as the errors analyzed in this study focus mainly on syntactic ones. |
| Impact on Society | The digital age is code-driven, thus error analysis in programming instruction will enhance programming ability, which will ultimately transform novice programmers into experts, particularly in developing countries where most of the software in use is imported. |
| Future Research | Researchers could look beyond novice or beginner programmers as codes written by intermediate or even advanced programmers are still not often completely error free. |


Keywords concept, Java, achievement level, error analysis, programming

## INTRODUCTION

Studies of taxonomies of novice programming errors have been of interest to many researchers and educators (Brown, Kooling, McCall, &Utting, 2014; Denny, Luxton-Relly, Tempero, & Hendrickx, 2011; Fitzgerald, Hanks, Lister, McCauley, & Murphy, 2013; Johansen,2015). Causes of such errors have also been investigated (Bringula, Manabat, Tolentino, & Torres, 2012; Kaczmarcyk, Petrick, East & Harman, 2010; Shah, Berges & Hubwieser, 2017). The ultimate aim in all of these has been to improve code writing and debugging skills. Such studies have always helped instructors to redefine their pedagogical approach with a view to addressing such identified errors and misconceptions. In doing this, teachers may need to create their own errors and chats in order to help the learners achieve long-term progress in program writing. Conceiving students' errors as valuable feedbacks drives instructors to apply remedial teaching based on the nature of their errors (Al-Saudi, 2013).Results from error analysis can also inform the design of courses, textbooks and also tools to target the most frequent (or hardest to fix) errors (Altadmri & Brown, 2015). The more educators understand about the nature of these errors and how students respond to them, the more effective teaching can be (Denny, Luxton-Reilly, & Tempero, 2012). However, most of the studies bordering on programming errors and misconception employed omnibus approaches, i.e., analyzing and taxonomizing errors without consideration of different programming concepts or domains. For example, Mow (2012) conducted

an exploratory study that investigated the most common errors students made in Java programming classes. Java codes written by undergraduates were analyzed for errors but the study did not take into consideration the topics that contained the majority of error types. Such an approach had often obscured some salient information because errors and misconceptions in one programming concept may be different from the other, thus the appropriate feedback needed for instructional improvement may be somewhat deficient. Jackson, Cobb & Carver (2005) checked and gathered syntactic errors of beginning programmers by an informal survey of current and former faculty members teaching the course. It was determined that there were discrepancies between the errors that the instructors had identified and the errors that the students were encountering; thus, it becomes clear that analysis of the code written by novice programmers themselves is the most reliable method in determining their errors. It is on this note that the current study seeks to analyze and identify categories of errors and misconceptions across different programming domains or concepts.

Apart from this, it is necessary to explore specifically how errors that novice programmers make vary based on their achievement level. High achieving students self-report having the easiest time learning the introductory programming topics. For example, in a quantitative analysis, high achieving and average students were more effective at debugging on average than low-achieving students (Rodrigo et al., 2014). This, however, does not address errors made across achievement level, which is logical to explore before assessing, debugging, identifying and correcting syntax errors; a challenge all novice programmers confront.

This focused analysis (i.e. based on concept and achievement level) is expected to provide a sound understanding of the learning process of each group of learners and their peculiarities across different programming concepts. In doing this, Java, one of the most popular programming languages globally, was used for the study. Java has been the second most popular language since its creation in the mid-90s (Tiobe Programming Index, 2017). While generating the index of the most popular programming languages, Tiobe employed variables such as the number of professional developers worldwide, training courses and third-party vendors in rating the popularity of the programming languages. The portability, scalability and large community of users of Java has led to its popularity. Also, Java shares a lot of core programming concepts with Python - another very popular language. It is believed that much of the syntax and concepts of the two languages are the same. It thus becomes imperative to conduct a study of this nature with a programming language such as Java knowing that findings from this study could also be of benefit in teaching Python. Specifically, the study will seek to identify the errors made by novice programmers in Java based on concepts. It will also determine the error types made by low achieving, average achieving and high achieving novice programmers in Java. This is in addition to identifying the errors made in Java programming across concepts based on the achievement level of the novice programmers. It is also necessary to explore possible misconceptions leading to the identified errors. Examining the relationships between errors in each of the concepts with the other concepts will help in determining whether errors in one concept predict errors in another concept. Similarly, committing a type of error in Java programming may suggest vulnerability to another particular type of error, thus it is also relevant to examine the relationship between each of the error types. This is the gap the study attempts to fill.

## METHODS

A sequential exploratory mixed-method design was adopted for the study. An intact class of 124 students studying 200-level computer science and engineering in a southwestern Nigerian university participated in the study. The study spanned one academic session. Introductory concepts in Java were instructed (with theory and practice) in the first semester after which the students were examined. A summative assessment based on the semester examination provided the basis for the categorization of students into "low achieving" (between 0 and 49 percent), "average achieving" (between 50 and 59 percent) and "high achieving" (between 60 and 100 percent) in Java programming. Twenty-six

students fell within the category of low achievers while 33 were in the average category and 65 constituted the high achieving cohort.

In the second semester, students were taken through practical sessions in Java and solutions to coding exercises were submitted through an online platform. The submissions covered concepts such as methods and classes, decision making, more object concepts and looping. Methods and classes include components such as "creating methods with no parameters, a single parameter and multiple parameters; creating methods that return values; class concept; creating a class; creating instance methods in a class; declaring objects and their methods; organizing classes." Decision making includes components such as "If, If…else, Nested If". More object concepts include "this reference, constructors, parameter sending, inheritance, using static variable, blocks and scopes." Looping consisted of "Loop structure, while loop, for loop, for-each loop, do … while loop, nested loops, use constant fields, use automatic imported, pre-written constants and methods".

Following Bringula et al. (2012), the submitted codes were analyzed for possible errors, using Quantitative Error Analysis. Identified errors were categorized into Invalid Symbols, Mismatched Symbols, Missing Symbols, Inappropriate Naming and Excessive Symbols. Invalid Symbols, according to Bringula et al. (2012) consist of errors such as "No period between class name and method name", capitalized keywords, Replacing (and) with <and> or [and] in output stream and else without if. Mismatched Symbols are a result of wrong curly braces, incorrect greater than or equal to sign, when the symbol cannot be found because of mismatched between the declared and used variable or undeclared variable. For Missing Symbols, errors include lack of semi-colon at the end of a statement, no close/open parenthesis on if condition and unclosed literals. Inappropriate Naming consists of bugs such as wrong casing of method names, inappropriate casing of class names and splitting a class name by putting a space while Excessive Symbols consist of excessive semi-colon, putting a period between the keyword, import and java packages and putting a semi-colon after the If-condition.

Quantitative Analysis was done, using descriptive statistics to determine error distribution based on concepts and achievements levels of beginner programmers. Pearson Product Correlation Coefficient was used to determine the relationship between error types. An interview was also conducted to elicit information on selected students (12 in all) on reasons or misconceptions leading to the errors made by them. Correlation is significant at the 0.01 level (2-tailed).

## RESULTS

In identifying the type of errors made by novice or beginner programmers based on concept, the code written by novice programmers in the course of laboratory practical were analyzed for errors. The errors were categorized into Invalid Symbol, Inappropriate Naming, Excessive Symbol, Mismatched Symbol and Missing Symbol, as previously described. Concepts and domains where the errors were committed were grouped into Decision Making, Looping, Methods and Classes and Other Object concepts (Table 1). Findings revealed that out of the 598 errors made, Missing Symbol was 195 (33%), least errors committed were Invalid Symbols (11.9%), Methods and Classes concept houses the highest number of errors 119 (35.8%) followed by Other Object Concepts (34%), Decision Making followed after (29.1%), with Looping (10.4%) housing the least number of errors.

In determining the error types based on achievement level, error types were grouped and categorized based on achievement level of the beginner programmers who committed them (Table 2). Findings revealed that novice programmers had the highest number of errors of Missing Symbol across all the achievement levels. Forty percent (40%) of Low Achievers, 28% of Average Achievers and 32% of High Achievers committed Missing Symbol errors.

To identify errors made across concepts based on achievement level, Table 3 shows that for Low Achievers, other Object Concepts constitute the most error prone (34%) followed by Methods and

Classes (24.4%). While for average and high achieving students, Decision Making (35%), and Method and Classes (32%) were the most error prone respectively.

In determining the relationships between each of the error types, Pearson product correlation coefficients were obtained between the error types (Table 4). Significant relationships exist between Missing Symbol and each of Invalid Symbol and Inappropriate Naming at 0.01 level. A significant relationship was also found between Missing Symbol and Mismatch Symbol at 0.05 level of significance. In addition to this, a significant relationship exists between errors made in Looping and each of the Decision Making and Other Object concepts at 0.05 level of significance. Errors made in Other Object concept and those made in Methods and Classes concept were significantly related at 0.01 level of significance (Table 5).

**Table 1. Error categorization and concept**

| | Invalid Symbol | Mismatched Symbol | Missing Symbol | Inappropriate Naming | Excessive Symbol | Total |
|---|---|---|---|---|---|---|
| Decision Making | 25 | 61 | 58 | 4 | 26 | 174 (29.1%) |
| Looping | 07 | 38 | 10 | 01 | 06 | 62 (10.4%) |
| Methods and Classes | 16 | 31 | 75 | 53 | 41 | 214 (35.8%) |
| Other Object Concepts | 23 | 03 | 52 | 20 | 48 | 146 (34.1%) |
| Total | 71 (11.9%) | 133 (22.2%) | 195 (32.6%) | 78 (13.0%) | 121 (20.2%) | 598 |

**Table 2. Error types and achievement level**

| | Invalid Symbol | Mismatched Symbol | Missing Symbol | Inappropriate Name | Excessive Symbol | Total |
|---|---|---|---|---|---|---|
| Low Achieving | 11 (8.7%) | 33 (26%) | 51 (40.2%) | 18 (14.1%) | 14 (11%) | 127 (21.2%) |
| Average Achieving | 19 (11.1%) | 45 (26.3%) | 48 (28.1%) | 14 (8.2%) | 45 (26.3%) | 171 (28.6%) |
| High Achieving | 41 (13.7%) | 55 (18.3%) | 96 (32%) | 46 (15.3%) | 62 (20.7%) | 300 (50.2%) |
| | 71 | 133 | 195 | 78 | 121 | 598 |

**Table 3. Concept-based errors and achievement level**

| | Decision Making | Looping | Methods and Classes | Other Object Concepts | Total |
|---|---|---|---|---|---|
| Low Achieving | 29 (22.8%) | 24 (18.9%) | 31 (24.4%) | 43 (33.9%) | 127 |
| Average Achieving | 59 (34.5%) | 17 (9.9%) | 55 (32.2%) | 40 (23.4%) | 171 |
| High Achieving | 86 (28.7%) | 21 (7%) | 130 (43.3%) | 63 (21%) | 300 |
| | 174 | 62 | 216 | 146 | 598 |

**Table 4. Relationships between error types**

| | | Invalid Symbol | Mismatch Symbol | Missing Symbol | Inappropriate Name | Excessive Symbol |
|---|---|---|---|---|---|---|
| Invalid Symbol | Pearson Correlation | 1 | | | | |
| | Sig. (2-tailed) | | | | | |
| Mismatch Symbol | Pearson Correlation | .218 | 1 | | | |
| | Sig. (2-tailed) | .050 | | | | |
| Missing Symbol | Pearson Correlation | .406** | .262* | 1 | | |
| | Sig. (2-tailed) | .000 | .018 | | | |
| Inappropriate Naming | Pearson Correlation | .195 | .050 | .287** | 1 | |
| | Sig. (2-tailed) | .081 | .656 | .010 | | |
| Excessive Symbol | Pearson Correlation | .117 | .206 | .043 | .052 | 1 |
| | Sig. (2-tailed) | .298 | .065 | .704 | .642 | |

*. Correlation is significant at the 0.05 level (2-tailed).

**. Correlation is significant at the 0.01 level (2-tailed).

**Table 5. Relationships between errors made by concepts**

| | | Decision Making | Looping | Method and Classes | Other Object Concepts |
|---|---|---|---|---|---|
| Decision Making | Pearson Correlation | 1 | | | |
| | Sig. (2-tailed) | | | | |
| Looping | Pearson Correlation | .234* | 1 | | |
| | Sig. (2-tailed) | .035 | | | |
| Method and Classes | Pearson Correlation | .077 | -.070 | 1 | |
| | Sig. (2-tailed) | .494 | .535 | | |
| Other Object Concepts | Pearson Correlation | .103 | .257* | .292** | 1 |
| | Sig. (2-tailed) | .360 | .020 | .008 | |

*. Correlation is significant at the 0.05 level (2-tailed).

**. Correlation is significant at the 0.01 level (2-tailed).

In identifying various misconceptions of novice programmers, twelve students from the intact class were randomly selected and interviewed on their perceived reasons for the identified errors committed by them. This was done by recalling their code and spotting the errors made. Participants were made to explain the reason(s) why they felt they made the spotted mistakes. This was in addition to a focus group discussion moderated by 3 of the researchers with the 12 selected students. The following responses were obtained and the implication of the misconceptions based on their responses were stated together with the errors that could arise from such misconceptions.

**Response 1:** *I give the void return type to both the getter and setter methods instead of void for setter and either int., string etc. For getter."*

***Implication:*** Class name can be declared wrongly which might affect the output from the object of the class.

**Susceptible Error(s):** Inappropriate naming error

**Response 2:** *We felt convention does not matter so far the code can run.*

***Implication:*** The omission of use of conventions sometimes leads to inappropriate naming error since there are rules guiding the naming of identifiers, variables and so on.

**Susceptible Error(s):** Inappropriate naming error

**Response 3…** t*he IDE we used for practicing were smarter, because it gave us hints on what went wrong and how it should be fixed but the editors we used in exams were not smart, they did not give us any hint on* errors.

***Implication:*** The use of smarter IDEs helps to curb the use of extra braces. The use of IDEs that are not smart allows student to make errors like excessive symbol, missing symbol and mismatched symbol since a computer with the aid of intelliSense will be able to identify these errors and help remove them automatically. Novice programmers will continue to commit such errors except when intelligent editors are used.

**Susceptible Error(s):** Excessive symbols, Missing symbols and Mismatched errors.

**Response 4***: Programming is like mathematics with precise formula and steps that must be memorized.*

**Implication**: Students therefore came to the examination practical with memorized code as the solution to problems that looked like previous ones. Memorizing code without necessarily understanding, leads to missing symbol, excessive symbols, mismatched symbol, inappropriate naming and invalid symbol errors. Missing symbol is because some portion of the code might be forgotten, excessive symbol is because an extra brace or symbol can be added, mismatched symbol can cause undeclared variable, unmatched curly braces among other problems. Also, it can cause improper naming of methods and classes.

**Susceptible Error(s)**: All forms of errors: Missing Symbols, Excessive Symbols, Mismatched Symbols, Inappropriate Naming and Invalid Symbols.

**Response 5:** *A student should be a man of one programming language.*

**Implication**: Programming with this mindset would lead to all types of error stated in this article. This is because students would lack the readiness to learn and adapt with the syntax and conventions of a new programming language being taught that was different from the previously learnt one. For example, for a student that had python as the first language, python is dynamically typed, in other words there is no need to declare a variable as a type before use. A python user will make the mistake of not declaring a variable in Java thereby leading to missing symbol error. Python uses indentation to separate code into blocks. However, Java uses curly brace. Indentation makes code readable with less chance of a missing brace. A python user may make a mistake of missing symbol when coding in Java. The implication of this is that of cognitive conflict or that disinterestedness in the new language leading to all forms of errors.

**Susceptible Error(s):** All forms of errors.

## DISCUSSION

The study determined that the Method and Classes concept is the most error-prone of all the concepts. This is probably because it serves as the gateway to Object-oriented Programming. Many of the enrollees were having their first Java programming experience, thus, methods and classes being the first concept in the course, the practical submissions in the concept were expectedly 'error-infected'. This is opposed to the findings of Adair and Jaeger (2011) who posited that the topic object with methods and classes was not found as being particularly difficult by students. However, Other Object Concepts which included inheritance and polymorphism were found to be somewhat difficult. Another reason for the prevalent error in method and classes was that some of the enrollees had previous (though little) exposure to other object-oriented languages, mainly python, with more liberal syntax. Java being a more 'disciplined' language is often characterized with stringent syntax thus, cognitive conflicts between the previously learnt language and the current one might have accounted for many of the errors in methods and classes. For example, in Java, every statement ends with a semicolon whereas this is not the case in Python. Also, due to the fact that Python does auto variable type assignment from the value stored in the variable, novice programmers might forget that Java is strict with variable type declaration and therefore omit the type, expecting the compiler to assign the type from the value. This is also the case for return type in Java method declarations.

Another critical issue is the readiness to learn a new language. Having had a bit of exposure to another language, one of the findings during the focus group discussion is that some of the participants believed that *a student should be a man of one programming language*. Hence, the non-amenability to Java syntax by such students.

Looping, however, housed the least number of errors. This is probably because even though Looping as a concept is of a higher cognitive hierarchy, errors in looping might not manifest in syntax form but rather as semantic or logic errors, which the error analysis in this study does not capture. For example, loop code written by a novice programmer can be syntactically correct but vulnerable to logic or semantic error as it can still run into an infinite loop. In other words, syntactically correct looping code does not necessarily inform adequate mastery of looping by novice programmers.

Missing symbols accounted for 195 (32.6%) of the errors, and invalid symbols (11.9%) constitute the least made error. It is also informative to note that missing symbols are errors that are likely committed at the earliest classes of programming. For example, the missing symbol errors (as shown in Table 1) reside largely within methods and classes giving an impression that bugs such as omission of semi-colons at the end of a statement and unclosed literals, which are beginners errors, are the preponderant mistakes. This finding is in agreement with that of Jadud (2006) when he reported that, out of 1926 errors encountered by students, more than half of all errors generated by students while programming are missing semicolons. Similarly, a study conducted by Mow (2012) summarized the top 8 errors to include variable not found (49.8%), class not found (5%), method not found (1.6%). Even though class not found and method not found may not necessarily imply missing symbols, the fact that variable not found accounted for half of the top 8 errors supported the finding that missing symbols accounted for the most frequent error types.

From this study, in particular, students were of the opinion that the IDE used by them for practicing were smarter than those used in the exam, as they did not give any hints on errors. This partly accounted for the increase in missing symbol errors. However, other object concepts such as parameter sending, inheritance, and constructors were also vulnerable to the error of missing symbols. This was largely attributed to inattentiveness. Bringula et al. (2012) found that in the field of programming, inattentiveness of students frequently yielded avoidable and simple errors. It is also relevant to observe that missing symbol errors cut across achievement levels.

The opinion held by some students that "*convention may not matter as long as the code can run*" may also be held across achievement levels. But low achievers committed more errors in Other Object concepts

while average achievers made more errors in Decision Making. High achievers made more errors in Methods and Classes. The explanation for this is that problems of Other Object concept with low achievers grew from Method and Classes. Other Object concepts and Method and Classes are of similar content with the former being of higher order cognition. It would seem that average achievers and high achievers had outgrown their method and classes problem. This is further corroborated with the fact that Pearson Correlation Coefficient showed that errors in Methods and Classes are strongly related with Other Object concepts. Errors in Missing Symbols have also been found to predict errors in each of Invalid Symbols, Inappropriate Naming and Mismatched Symbols. Thus, Missing Symbols form the kernel of almost all the errors.

A major limitation of the study was that the Array concept that was scheduled to be part of the course content could not be examined. The feedback from the study was that if the Array concept were included it would have made the study more robust. Also, the errors analyzed in this study were mainly syntactic. Studies are needed that would enquire and analyze logic and semantic errors of novice programmers.

## IMPLICATION FOR CLASSROOM PRACTICE

Generally speaking, students come to the beginner programming class with many misconceptions, which the first lessons should address. For example, early lectures should first address the art of programming and correct the erroneous beliefs that code is like mathematical steps that one can memorize. This misconception, as stated earlier, could lead to all types of errors.

To promote better classroom practice in the teaching of java programming, findings for the study suggests instructions to students should be based on achievement level. This is because errors have been found to be achievement-based. In most undergraduate computer science curricula, Java starts with a theoretical course with practical sessions gradually introduced, and later course(s) may be largely laboratory based. The first Java programming course should assist in categorizing students into low, average and high achieving cohorts. It should also be used to compile error logs for each achievement level. Programming instruction in the later courses should target prevalent errors for each achievement group having regard also to the concepts that have mostly specific types of error. This will enhance programming teaching effectiveness and better learning outcomes. In a situation where it becomes impossible to separate into achievement groups continuously, it would be helpful to at least separate the low achievers at the beginning (during methods and classes practical sessions) for focused attention. It was observed that it takes low achievers more time to overcome the beginner errors. Their average and high achieving counterparts overcome the typical beginner errors as the class progresses.

In addition to this, learning Java programming should be done with an unintelligent editor. Intelligent editors (though easier to learn programming with) will return less bugs and may give a false impression of coding mastery for beginners. Learning is thus enhanced with an unintelligent editor as learners are forced to think, reason and spot on their own any bugs in the code. The editor used for laboratory sessions should also be used during examination for consistency and assessment validity. This approach will largely minimize missing symbol errors, which is the most prevalent.

## CONCLUSION

This study analyzed error types and patterns in Java programming based on fundamental concepts of Methods and Classes, Decision Making, Other Object concepts and Looping of beginners programmers at different achievement level. Missing symbols were found to be the commonest type of error and invalid symbols constituting the least. The Method and Classes concept had the highest number of errors with Looping having the least. Error types were also found to be the same across ability levels. However, expectedly low achieving students had more challenges writing bug free code in Other Object concepts. Programming assignment instructions in Java should take into considera-

tion the prevalent errors within the achievement cohorts as well as the concepts that are prone to a specific type of error.

## REFERENCES


Adair, D., & Jaeger, M. (2011). Difficulties in teaching and learning the Java programming language. *Proceedings of the 17th International Conference on Engineering Education*, *Northern Ireland*, 21-26.

Al-Saudi. (2013). Error analysis and spelling mistakes of EFL learners at Tafila Technical University: A case study. *Frontiers of Language and Teaching, 4*, 99-107.

Altadmri, A. T., & Brown, N. C. C. (2015). 37 million compilations: Investigating novice programming mistakes in large scale student data. *Proceedings of the 46th ACM Technical Symposium in Computer Science Education, Kansas City, Missouri,* 522-527. https://doi.org/10.1145/2676723.2677258

Bringula, R., Manabat, G. M., Tolentino, M. A., & Torres, E. (2012). Predictors of errors of novice Java programmers. *World Journal of Education*, *2*(1), 3-15. https://doi.org/10.5430/wje.v2n1p3

Brown, N. C. C., Kölling, M., McCall, D. T., & Utting, J. (2014). Blackbox: A large scale repository of novice programmers activities. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, *Atlanta, Georgia, USA*, 223-228. https://doi.org/10.1145/2538862.2538924

Denny, P., Luxton-Reilly, A., & Tempero, E. (2012). All syntax errors are not equal. *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education, Haifa, Israel,* 75-80. https://doi.org.10.1145/2325296.2325318

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*, 208-212. https://doi.org/10.1145/1999747.1999807

Fizgerald, S., Hanks, B., Lister, R., McCauley, R., & Murphy, L. (2013). What are we thinking when we grade programs? *Proceeding of the 44th ACM Technical Symposium on Computer Science Education, Denver, Colorado, USA,* 471-476. https://doi.org/10.1145/2445196.2445339

Jackson, J., Cobb, M., & Carver, C. (2005). Identifying top Java errors for novice programmers. *Proceedings Frontiers in Education 35th Annual Conference, Indianopolis, IN, USA.* https://doi.org/10.1109/FIE.2005.1611967

Jadud, M. (2006). *An exploration of novice compilation behaviour in BlueJ.* Unpublished Doctoral Dissertation, University of Kent, UK.

Johansen, M. J. (2015). *Errors and misunderstandings among novice programmers: Assessing the student not the program.* Unpublished M.Sc Thesis, University of Oslo, Norway.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying students misconceptions of programming. Proceedings of the 41st technical symposium on computer science education, March 10-13, 2010, Milwaukee, Wisconsin, USA. https://doi.org/10.1145/1734263.1734299

Mow, I. C. (2012). Analysis of student programming errors in Java programming courses. *Journal of Emerging Trends in Computing and Information Sciences*, *3*(5), 739-749. Retrieved from www.cisjournal.org

Rodrigo, M. M. T., Andallaza, T. C. S., Castro, F. E. V. G., Armenta, M. L. V., Dy, T. T., & Jadud, M. C. (2014). An analysis of Java programming behaviours, affect, perceptions and syntax errors among low-achieving, average and high-achieving novice programmers. *Journal of Educational Computing Research 49*(3), 293-325. https://doi.org/10.2190/EC.49.3.b

Shah, P., Berges, M., & Hubwieser, P. (2017). Qualitative content analysis of programming errors. *Proceedings of Fifth International Conference on Information and Education Technology, Tokyo, Japan*, 161-166. https://doi.org/10.1145/3029387.3029399

TIOBE Quality Indicator. (2017). Available at http://www.tiobe.com

## BIOGRAPHIES

**Philip Olu Jegede** is Professor of Computer Science Education in the Institute of Education, Obafemi Awolowo University, Ile-Ife, Nigeria, the chair he has been occupying since the year 2009. He is a holder of B.Sc. and M.Sc. degrees from University of Lagos, Nigeria. He later obtained M.Ed. and Ph.D. degrees from Obafemi Awolowo University, Ile-Ife, Nigeria. He was Director, Institute of Education of Obafemi Awolowo University between 2008 and 2010 and Dean, Faculty of Education of the same university between 2011 and 2016. His research interest has focused largely on computer studies pedagogy and presently on computer programming education. Before his university teaching appointment, he had lectured in a College of Education and a Polytechnic School.

**Emmanuel A. Olajubu** is a graduate of Computer Science (*with Economics*) from the Department of Computer Science & Engineering, Obafemi Awolowo University, Ile-Ife. He holds research degrees M.Sc. and Ph.D. in Computer Science from the same Department (2003 and 2008 respectively). He is also a member of Nigerian Computer Society (NCS), Computer Professional Registration Council of Nigeria (CPN) and International Association of Engineers (IAENG). He has over forty-five published articles in reputable journals and referred conference proceedings. Research interest is the area of distributed systems and network security. He is the current Acting Head, Department of Computer Science & Engineering, Obafemi Awolowo University.

**Adekunle O**. **Ejidokun** had his Bachelor's degree (B.Tech) in Computer Science from Ladoke Akintola University, Ogbomoso, Oyo State and his Master's degree in Computer Science from Obafemi Awolowo University, Ile-Ife, Osun State. He is a member of Nigerian Computer Society (NCS) and Computer Professional Registration Council of Nigeria (CPN). He is currently a Ph.D. student of Obafemi Awolowo University, Ile-Ife, Nigeria. He is currently working on community detection in real-world social network for his Ph.D. His research interests are Social Network Analysis (SNA), Information Storage and Retrieval and Computer Education.

**Isaac Oluwafemi Elesemoyo** is a doctoral student of the Department of Computer Science and Engineering, Obafemi Awolowo University, Ile-Ife, Nigeria. He received his M.Sc. and B.Sc. In Intelligent Systems Engineering and Computer Engineering respectively from Obafemi Awolowo University. His research interests are Computational linguistics and Computer Education. He is currently an Assistant Lecturer at Elizade University, Ilara-Mokin, Nigeria.