Volume 4, Issue III, March, 2025, No. 62, pp. 793-801
Submitted 1/3/2025; Final peer review 30/3/2025
Online Publication 6/4/2025
Available Online at http://www.ijortacs.com

CRITICAL STUDY OF THE OOP, FEATURES, IMPLEMENTATION AND ATTRACTED STRENGHTS TO JAVA PROGRAMMING LANGUAGE

P.E. Kekong [1*], Ewhenke B. Kekong[2], Michael E. Ekpo[3],

[1]Federal Univerity of Health Sciences, Otukpo Benue State, Nigeria
[2]University of Uyo teaching hospital, Uyo Akwa Ibom State. Nigeria
[3]Department of Computer Science, Ebonyi State University, Abakaliki, Nigeria

[1*] piuskekong2019@gmail.com, [2]bettyreal2014@gmail.com, [3]ekpom550@gmail.com,

Corresponding Author's Email and Tel: piuskekong2019@gmail.com ; +234 808 521 1771

Abstract

The Java programming language's Object-Oriented Programming (OOP) characteristics are thoroughly examined in this paper, along with their benefits and drawbacks. The cornerstone for creating reliable, scalable, and maintainable software systems is Java's commitment to fundamental OOP principles like encapsulation, inheritance, polymorphism, and abstraction. Modularity and data security are improved by encapsulation, and the effective creation of complex systems and code reuse are encouraged by inheritance. Flexibility and extensibility provided by polymorphism allow for simplified code management and dynamic method binding. Although abstraction may lead to an increase in complexity and development overhead, it makes it easier to create modular systems and clear interfaces. While these OOP capabilities add a lot of value to Java, the study also notes that there are drawbacks, including performance overhead, debugging difficulty, and the possibility of developing tightly connected systems. The results highlight how crucial it is to apply OOP principles in a balanced way in order to fully utilise Java's potential for creating efficient and maintainable software solutions. The purpose of this analysis is to shed light on how Java's OOP features can be used wisely to create software that is more reliable and efficient.

Keywords: OOP; Java; Encapsulation; Polymorphism; Inheritance; Abstraction

1. INTRODUCTION

Algorithm design, coding, testing, debugging, and implementation are all part of programming. Problem-solving skills and computer programming are closely related, but at the foundational level, students are not expected to solve complex problems; instead, they should begin by learning the fundamentals in order to develop a higher level of cognitive understanding and possibly a better understanding of more advanced programming concepts. Finding various ways to present and solve programming challenges is part of the challenge of helping students grasp programming principles. Making the switch from one programming paradigm to another, like procedural to object-oriented programming (OOP), can be difficult because there are a lot of concepts that overlap [8].

Programmers quickly discovered that groups of related functions that handle a large amount of data into logical groups makes a programme easier to read and run. This kind of combination of data and functions is called class and object grouping. Additionally, creating programmes with classes is known as object-oriented programming [11].

Since the objects in the OOP problem domain are related to real-world items, the OO paradigm is thought to be a more natural domain to deal with [4]. However, it can be very tough and demanding for students to comprehend and relate the fundamental ideas of object-oriented programming (OOP)such as classes, objects,

attributes, methods, method passing, inheritance, polymorphism, and encapsulation to real-world situations [21].Object-oriented programming is designed to apply real-world ideas in programming, such as inheritance, hiding, and polymorphism. The main objective of OOP is to connect the functions that manipulate the data with the data itself so that only that function and no other part of the code can access the data.

Eliminating some of the procedural approach's shortcomings served as the primary driving force for the development of the object-oriented approach. OOP restricts the free flow of data across the system and views it as a vital component of programme development. It safeguards data from inadvertent alteration by external functions and strengthens the link between the data and the functions that use it. With OOP, an issue can be divided into several things known as objects, and data and functions can then be built around these objects. Figure 1 illustrates how data and functions are arranged in object-oriented programmes. A function that is linked to an object is the only way to access its data. Nonetheless, Nzerue-Kenneth et al. [16] state that the function of one object can access the function of other objects.

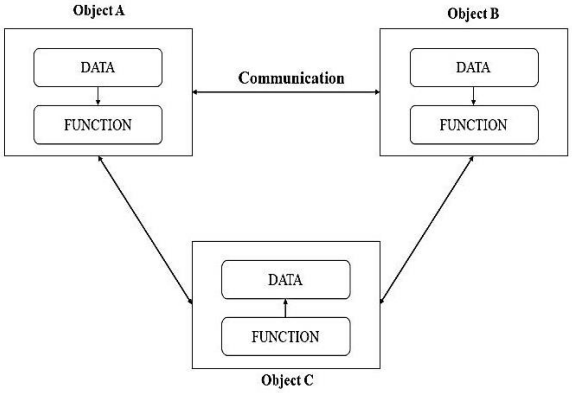

Figure 1: Organization of data and function in OOP

2. LITERATURE REVIEW

[20] researched on the basic concept of object-oriented programming. Object-oriented programming, or OOP, has become more widely used in software because of its potential to develop the software business and advance software engineering. In order to study object-oriented programming in depth, it is necessary to be aware of some crucial features. This study covers the concept of object-oriented programming, including its features, benefits, and drawbacks, as well as the constructor and destructor concepts. Software development is shown to rise with the use of features like classes, objects, encapsulations, polymorphism, inheritance, and abstraction.

[16] presents a detailed study on the features of object-oriented programming in python. The paper claims that OOP was created in order to get around the drawbacks of procedural programming. Because new objects in Python are formed by inheriting properties from existing ones, OOP makes maintenance and change of existing code easier. Because OOP enables the developer to safeguard Python's important data pieces, it is more secure. One way to achieve this is to limit access to data to only those methods that are part of that specific object; this is known as abstraction. Furthermore, OOP's programme modularity makes it easier to generate new data items from pre-existing ones, which reduces complexity. Adapting object-oriented programming is made simple by this idea.In order to create complete and well-designed Python software solutions, it is essential to grasp OOP thoroughly.

[18] surveyed on the concepts of object oriented programming. This examination delves deeply into several OOP topics that are fundamental to object-orientation. In this review article, the concepts, significance, and uses of object-oriented programming (OOP) are extensively discussed. For building complex systems, the modular, reusable, and maintainable OOP design style is perfect. It uses inheritance to promote code reuse, polymorphism to allow objects to exhibit alternative behaviours, abstraction to lessen complexity, and classes to contain information and functions. Different aspects of inheritance and polymorphism are implemented by numerous popular object-oriented programming languages. We conclude that in order for them to achieve OOPs features, a great deal of work needs to be done to establish a medium ground.

[14] Researched on the concepts of object-oriented programming. This study on Object Oriented Programming has been the fields of the Software Development. This study covers some of the most well-known and extensively used object-oriented programming languages as well as the principles and ideas behind OOP. The distinction between procedure-oriented and object-oriented programming was also covered. Benefits and Drawbacks of Object-Oriented Coding Despite a few drawbacks, object-oriented

programming (OOP) has many benefits and is effectively used in a wide range of software industry domains, including the internet, robotics, gaming, and scientific applications. Encapsulation, polymorphism, inheritance, abstraction, and classes are examples of OOP elements that can be used to construct real-time projects and products more effectively.

[1] researched on the gauge of OOP for student's learning performance, impacts on normalized learning and perceived motivation with Serious Games (SG). The goal of this study is to create a successful SG prototype that will help students overcome their obstacles and misconceptions when studying OOP. To compare the performance of the students in the experimental group who engage with the created game and the control group who receive instruction using the conventional approach, an experimental evaluation was conducted. The primary conclusion drawn from the experimental assessments is that the experimental group outperforms the control group in terms of performance. Normalised Learning Gain (NLG) for the experimental group is considerably larger than that of the control group ($p < 0.005$, paired t-test).According to the evaluation study's findings, the produced prototype's perceived motivation scored highest for attention (3.87) and relevance (3.66) subcategories on the Instructional Materials Motivation Survey (IMMS) 5-point Likert scale.

[18] presents the study on benefits of applying OOP for objective evaluation of vehicle dynamic performance in concurrent simulations. This work focuses on a C++ simulation environment that has been customised to take advantage of object-oriented programming features. Concurrent simulations of automobiles with various attributes, including mass, tyres, engines, suspension, and gearbox systems, are the goal of the framework that is being described. We used a modular and hierarchical representation in the suggested simulation framework. A full-vehicle model with 14 degrees of freedom (DOF) is used to model vehicles. It can capture the dynamics of the vehicle and is supplemented by a series of scalable-detail models for the other subsystems, including the engine, steering system, and tyres. Additionally, the use of autonomous virtual drivers is suggested in this paper for a more impartial assessment of the dynamic performances of vehicles. Additionally, in order to gauge the effectiveness of our simulator architecture and the degree of concurrency attained, the researchers built a benchmark that could measure how performance scaled in relation to the number of cars used in a given simulation. The study concludes by reporting on the scalability of the suggested simulation environment as a result of a variety of distinct and variable driving scenarios.

[22] Presents an educational software quality assessment tool for java called SQMetrics. The SQMetrics application's primary goal is to meet academic or research demands by offering the convenience of measuring small code, rather than to compete with currently available commercial products. Within this context, the programme is easily used by teachers to assess students' Java projects, which are assessed according to quality, as well as by software engineering students to make measurements and comparisons in their projects with the goal of improving the indicators measured. Since feedback is crucial for helping students in software engineering education to better their work, SQMetrics can be used to offer unbiased comments on the calibre of software projects. Furthermore, tests and analyses were conducted to determine the effectiveness of SQMetrics as an Android Antenna Tool (AAT) for evaluating the quality of Java code. The software tool can be a trustworthy and accurate assistance to teacher grading, as evidenced by the results, which revealed a positive association between instructor rating and the overall quality index generated from the programme.

3. RESEARCH METHODOLOGY

The investigation will be conducted using a qualitative research technique. The goal of qualitative research methods is to gather information that is often difficult to articulate in non-numerical form. A certain amount of interpretation on the side of the researchers is usually involved when collecting data by observation, coded survey or interview responses, and other techniques. To help with the interpretation of their findings, researchers may use a range of qualitative approaches in a single study in addition to a theoretical or critical framework [5].

3.1 THE CONCEPT OF OBJECT-ORIENTED PROGRAMMING

A novel approach to using computers to solve issues is object-oriented programming. The idea of an object starts to gain traction around 1970 among computer language researchers. According to [2], an object is a set of code and data that is intended to mimic a real-world or abstract entity. A programming approach known as "object-oriented programming" links data structures to a group of operators that perform operations on them. An instance of such a thing is referred to as an object in OOPs parlance. Relationships between objects are prioritised over implementation specifics. When implementation details are kept hidden inside an object, the user becomes more focused on how an item interacts with the system as a whole rather than how the object is implemented [7].

The fundamental method of programming in object-oriented programming languages is known as the "Object Oriented Programming Paradigm." The four key strengths of the paradigm are "abstraction," "encapsulation," "inheritance," and "polymorphism" [14], These OOP characteristics are implemented in a largely similar way by all OOP languages. OOP programmes comprise a collection of relational classes and objects, or data, that provide code flexibility and reusability through inheritance and polymorphism, respectively [20]. A large, complex project can be broken into a number of relational classes in order to achieve low production time and cost. This reduces the complexity of the project operations and allows for the effective management of large, complex projects in real time. OOP makes it easier for consumers and developers to conceal complex information [23].

Software is included in the "Classes" idea. The foundation of all object-oriented programming (OOP) is "Objects." Specifically, OOP focuses on modelling classes and objects using "Objects."One of the key ideas of object-oriented programming (OOP) is "Message Passing," which refers to the use of all project objects for message sending and receiving. OOP programmes use data and code, which allows for code reusability and stability, allowing for the quick development of large, complicated projects [24].

A. Features of OOP

The following properties are defined by the Object Oriented Programming System (OOPS). Any of the following features could be implemented by any OOP language, whether old and new [14]. Figure 2 presents the features of OOP.

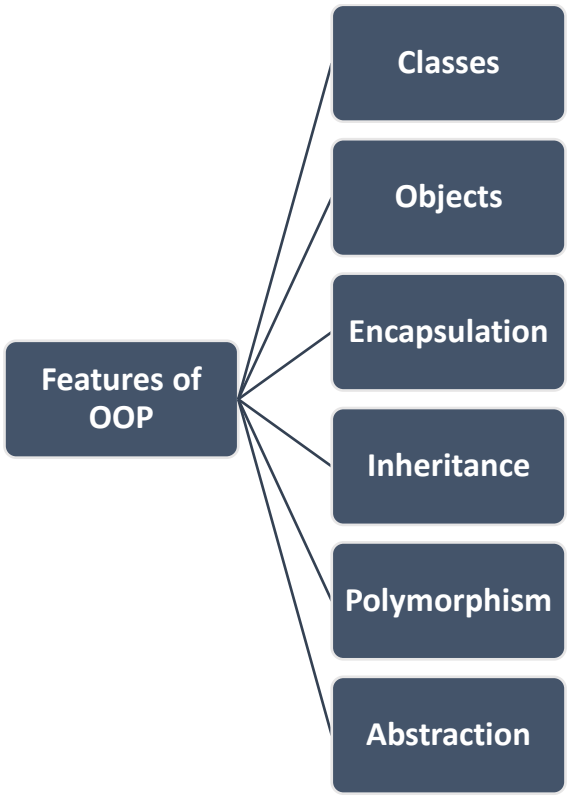

Figure 1: Features of OOP

(a) Classes

The foundation of any OOP programme is a class. Classes are made up of both code (functions) and data (variables). Classes protect the data and code from unauthorised users' misuse and outside influence. Class is a hypothetical concept that does not exist in reality because it is a logical programming construct. A few unique objects are present in every class of an OOP programme and are used for "Message Passing." Message passing refers to the ability of objects to manipulate data and functions; in other words, objects can send and receive messages from code (functions). Message passing is among the core ideas of object-oriented programming. Additionally, classes enable other classes to derive their variables and functions, paving the way for hierarchical abstractions, which designate which class or classes should execute first.

The fundamental component of implementing OOP concepts is the classroom [6].

(b) Objects

The core component of every OOP programme is its objects. An object's behaviour, data, and state (attributes) together define its functionality. An instance of a class is called an object. In contrast to classes, objects are actual physical entities that exist in the real world. An employee could have attributes like name, department, pay, and empno, for instance. as well as their actions when they work, take time off, etc. An employee is an item in this sense, with certain characteristics and behaviours. Items may contain confidential information that should only be viewed by those with permission. Unauthorised individuals should not have access to sensitive information. Many OOP languages have access specifier mechanisms that can be used to restrict data access. "Message Passing" is another crucial feature of objects. It indicates that messages from functions can be sent and received by the objects [16].

(c) Encapsulation

One of the fundamental ideas of OOP is encapsulation. A method of connecting code (functions) and data (variables) is called encapsulation. The fundamental unit of encapsulation in the object-oriented programming paradigm is the class, which is the fundamental unit of a programme. Data and code are bound together by a class, which also shields them from outside interference and unauthorised use. Encapsulation, also known as information hiding, is frequently used to shield a class's internal state from outside interference. By doing so, we may regulate access to the internal state of class instances while simultaneously concealing confidential information. Encapsulation will make it easier to demonstrate the relationships between the data and class methods in integration by ensuring their completeness and integrity.

(d) Inheritance

One key idea in OOP is inheritance, which allows one class to inherit the characteristics of another class. Stated otherwise, a class has the ability to inherit the characteristics and actions of another class. Functions and attributes from the base class (parent class) will be derived into the derived class (child class) via inheritance. The primary purpose of inheritance is to enable code reuse. By giving a derived class, or child class, access to its base class's properties and functions, inheritance reduces execution time and minimises errors. Utilising the qualities and capabilities again Make programming simple and allow users to reuse code from our current class in our new Derived class with great ease. There is a "is-a" connection between inheritance and derived classes. A class can also inherit its properties and functionalities from many classes by using inheritance. The other inheriting classes will work differently if modifications are made to the inherited classes [8].

(e) Polymorphism

Polymorphism is another crucial OOP concept. Users can more easily create a shared interface for a collection of related activities when polymorphism is used. Put differently, polymorphism permits the definition of a single Name (object) and several forms (functions). A function can be implemented differently in base classes and derived classes and take on different forms thanks to polymorphism. Due to code simplification, OOP programmes that implement polymorphism will run quickly. For instance, in OOP languages, implementing a stack data structure requires the creation of a single common interface (object), which can be applied to a variety of forms and data kinds. In non-OOP languages, however, this is not feasible [8].

(f) Abstraction

Above all other aspects of OOP is abstraction. Users can control an object's complexity thanks to abstraction. Abstraction conceals from the outside world the specifics of an object's implementation, concentrating instead on the essentials of the item. Additionally, it enables users to interact with those intricate object components through their interfaces. For instance, let's say that "Car" As an object, a car merely displays interfaces such as steering, breaking, and lighting to the user (driver), hiding all other sophisticated operations. Put another way, an automobile item grants the user access to the interfaces of its complicated parts while concealing from them the functionality of those elements. The distinction between encapsulation and abstraction is that the former focuses only on the pertinent details of the object and conceals from the

user any superfluous complexity, while the latter binds and conceals the data and functions into a single unit, enveloping it in a protective layer that guards against unauthorised access [11].

3.2 Programming Languages for the Implementation of OOP

Programming languages that are object-oriented have a long history dating back to the 1960s. Little Talk, Simula 67, and Simula are the original OOP languages. Simula 67 served as the model for the development of Small Talk. However, Small Talk is more widely recognised as a completely OOP language, meaning that everything in Small Talk is an object. Later, C++ developed into an OOP language in the 1980s. Basic programming constructs from the C language are included in C++. For this reason, C++ was once known as "C with Classes." The 1990s saw the beginning of the Golden Age of Object-Oriented Programming languages. One aspect of OOP languages that has evolved in the software industry is the Java language, which was invented in 1995.It's because Java Language's programming constructs are both powerful and easy. Then, on the OOP stage, Ruby and C# have become major roles [4].

Java is now a well-known multi-paradigm programming language, at last. Java is quite popular due of its adequate implementation of OOP elements and its simplicity. The purpose of Java development is to satisfy real-world software needs. Some or all of the features of object-oriented programming are implemented by each of these programming languages. For instance, because everything in small talk is an object, practically all OOP characteristics are implemented in small talk [8]

3.3 A REVIEW OF JAVA PROGRAMMING LANGUAGE

James Gosling created the Java programming language in 1995. Java is a class-based, object-oriented, high-level programming language that has gained a lot of popularity. It is intended to activate the "Write Once, Run Anywhere" (WORA) function, which permits Java code that has been built to run on any platform that supports Java without the need for additional compilation [12]. Java is a popular choice for developing a wide range of programmes, including mobile apps, web apps, desktop apps, and games. It is noted for its simplicity, portability, and platform independence. Because Java is turned into bytecode, which can operate on any platform that supports the Java Virtual Machine (JVM), one of its primary features is its platform freedom. This eliminates the need for developers to worry about platform-specific details and allows them to create code once and run it on any platform [9]. Furthermore, a lot of businesses and academic institutions utilise Java, and it is essential for many business applications.

A. Features of Java Programming Language

Programming languages like Java are popular because of their reputation for performance, platform freedom, and security. Java is a programming language for computers that may be used for many different things, including games, desktop computing, back-end and android development, and numerical computing [3]. The main characteristics of the Java programming language are covered in this essay. Among the attributes of programming languages in Java are:

i. **Object Oriented:** In Java, everything is shown as an object. Software can be organised by anyone through the combination of many entity types that include data and functions. Because Java incorporates OOP, it is now widely used in the creation of complex software systems, especially in enterprise settings where scalability and modularity are important factors.

ii. **Simple:** Because of its clear, basic, and understandable syntax, Java is a very straightforward language to learn. Sun Micro system claims that because the syntax is derived from C++, developers with knowledge of C++ or related languages will find it easy to use. Additionally, it comes with a vast library of pre-built classes and methods that offer many common functions required by developers while creating applications. Another reason it is regarded as a simple language is its platform independence, which allows programmers to create code once and have it execute on any machine that supports Java. Generally speaking, Java's ease of use makes it a popular option for both novice and seasoned developers, which helps to explain the language's popularity and wide acceptance.

iii. **Secured:** Given that it created a virus-free system with no explicit pointers and virtual machine box operation, Java is most recognised for its security. Furthermore, Java features a class loader that isolates the class package from both the local file system and imported network sources, hence enhancing security. The security manager determines what local disc resources a class can use, while the bytecode validator checks code fragments for any illegal code to guarantee secure access to the right objects.

iv. **Platform Independent:** Because it relies on software-based technologies to function atop hardware-based platforms, the Java platform stands out from other platforms. There are two parts to the platform: The class libraries and other resources required to run a specific Java programme are provided by the runtime environment, which operates on top of the computer's operating system software. – An easily accessible method of extracting and sharing data within and between organisations is through APIs (Application Programming Interfaces).

v. **Robust:** Java's effective memory management system makes it a programming language that is regarded as resilient and safe. The absence of pointers in this technique reduces security vulnerabilities, which is one of its advantages. It also uses the Java Virtual Machine for automatic garbage collection, which removes unused objects from a Java application. These all contribute to Java's strength.

vi. **High-Performance**: Java is becoming more and more popular for High-Performance Computing (HPC) because of its attractive features for multi-core cluster architecture programming, particularly its support for multithreading and built-in networking. Furthermore, the Java Virtual Machine (JVM) has become even more appealing in this context due to its continuous performance improvement.

3.4 STRENGTHS ATTRIBUTED TO JAVA'S OOP IMPLEMENTATION

The Java programming language has all the capabilities required to be used as an Object-Oriented programming paradigm, as section 5 illustrates. Its merits include the following, which make it suitable for OOP implementation: Figure 3 presents the strength of OOP.

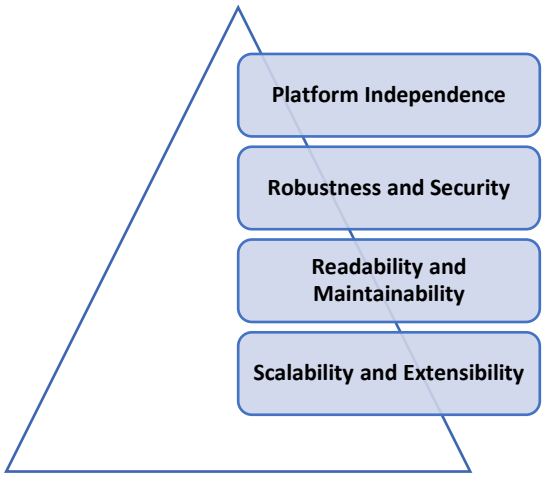

Figure 3: Strength of Java OOP

a. Platform Independence

OOP features in Java help to considerably support the "write once, run anywhere" idea. Encapsulation, inheritance, and polymorphism are three techniques that facilitate the creation of modular code that is readily portable between platforms.

b. Robustness and Security

Encapsulation protects data integrity and stops unwanted access, which improves the security and resilience of Java applications. Java's OOP model integrates seamlessly with its exception handling features, which further aid in the development of robust programmes.

c. Readability and Maintainability

Because Java strictly adheres to OOP concepts, the code is easier to read and maintain. Large codebases are easier to manage and update thanks to features like inheritance and polymorphism, which reduce code redundancy.

d. Scalability and Extensibility

Java's OOP features, including polymorphism and abstraction, make it easier to create scalable and expandable programmes. Large systems can evolve over time by adding new functionality with little to no changes to the old code.

Java's implementation of OOP concepts offers a strong base upon which to construct scalable, maintainable, and reliable programmes. Although it has many advantages, like flexibility, modularity, and code reuse,

it also has drawbacks in terms of complexity and performance. It is essential to comprehend these characteristics and their ramifications in order to use Java in software development efficiently.

4. CONCLUSION

This paper highlights the benefits and drawbacks of implementing Java's Object-Oriented Programming (OOP) capabilities through a rigorous analysis. Java's capacity to create reliable, scalable, and maintainable programmes is highlighted by its adherence to fundamental OOP concepts like encapsulation, inheritance, polymorphism, and abstraction. The language's reputation as a dependable and adaptable tool for a variety of applications, from enterprise-level systems to mobile apps, is largely due to these qualities, each of which has substantial benefits.

Java encapsulation facilitates the creation of well-structured, error-resistant programmes that are easier to maintain, hence promoting modularity and data security. It may, however, also result in development overhead and limit flexibility in situations involving rapid development. Because inheritance follows the DRY (Don't Repeat Yourself) concept, it makes code reusability easier and speeds up the development process. Still, it can lead to highly connected systems that are more difficult to manage and prone to problems such as the fragile base class problem. Because polymorphism makes code more extensible and flexible, it facilitates the simpler integration of new features and allows for more generic processing of objects. However, because method dispatch is dynamic, it might cause performance cost and make debugging more difficult. Abstraction facilitates the definition of unambiguous interfaces and encourages a high degree of modularity, which makes systems simpler to comprehend and alter. However, in large systems, it can also result in a lot of boilerplate code and add complexity.

Because of its effective use of these OOP concepts, Java is a popular choice among developers globally due to its usefulness. The language's attractiveness is further enhanced by its support for platform independence. Java's OOP characteristics are not without drawbacks, despite these benefits. Notable concerns include the performance overhead related to features like dynamic method dispatch and the complexity brought about by large abstraction layers and deep inheritance hierarchies. These difficulties demand considerable thought to be taken into account in the design and development stages in order to guarantee that the advantages of OOP are optimised without suffering major shortcomings.

To sum up, Java's implementation of OOP features offers a strong base for creating software that is reliable, manageable, and scalable. Through the proper use of encapsulation, inheritance, polymorphism, and abstraction, developers may design robust and versatile systems. Even with these obstacles, software solutions can be extremely successful if these functionalities are well understood and used sparingly. In order to get the best possible software design and performance, this study emphasises how crucial it is to apply OOP principles in Java in a balanced and careful manner.

5. AUTHOR'S BIOGRAPH

Kekong Pius Ekwo is a Lecturer in the Department of Computer Science at the Federal University of Health Sciences, Otukpo, Benue State, Nigeria. He is a registered member of the Computer Professionals Registration Council of Nigeria (CPN). His areas of academic and research interest include artificial intelligence, cybersecurity, adaptive control systems, and embedded systems. He has authored and co-authored several peer-reviewed publications in national and international journals. He is currently pursuing his Ph.D. in Computer Science at Ebonyi State University, Abakaliki, Nigeria, where his research focuses on the application of neural networks in satellite attitude control and intelligent systems design.

Ekpo Michael Ernest is a doctoral student in the Department of Computer Science at Ebonyi State University, Abakaliki, Nigeria. His research interests span across machine learning, cyber security, intelligent decision support systems, and data-driven risk assessment techniques. He has contributed to academic research in adaptive learning

systems and is actively involved in collaborative projects focused on the use of AI in enhancing security and control systems in health technology.

Adams is currently pursuing his Ph.D. in Computer Science at Nasarawa State University, Keffi, Nigeria. His research explores advanced topics in artificial intelligence, software engineering, and Internet of Things (IoT) applications for smart systems. He is particularly interested in the integration of AI-driven models for real-time monitoring, automation, and security enhancement. He has participated in national academic workshops and is involved in ongoing research on intelligent system control for autonomous platforms.

6. REFERENCES

1) Abbasi S., Kazi H., Kazi A., Khowaja K., & Baloch A., (2021) Gauge Object Oriented Programming in Student's Learning Performance, Normalized Learning Gains and Perceived Motivation with Serious Games. Information 2021, 12, 101. https://doi.org/10.3390/info12030101
2) Bhusari, C. S., Vaz, S. A., &Angne, S. (2019). Concepts of object-oriented programming. International Journal of Scientific Research, 8*(6), 1-4. https://doi.org/10.56726/IRJMETS31010
3) Deepali (2023) Features of Java Programming Language. https://www.interview-bit.com/blog/features-of-java/
4) Elliott E., (2023) "The Future of Programming: AI and Interface-Oriented Languages", written by Javascript Scene
5) Fulton Library (2024) Research Methodologies. Qualitative Research Methodologies - Research Methodologies - Research Guides at Utah Valley University (libguides.com) Accessed June 2024
6) Geek for geeks (2024) Object oriented programming in C++. Object Oriented Programming in C++ - GeeksforGeeks Accessed June 2024
7) Ghodke, R. B., & Gaikwad, G. D. (2023). Basic concept of object-oriented programming (OOP). International Research Journal of Modernization in Engineering Technology and Science, 5*(9), 1-6. https://doi.org/10.56726/IRJMETS31010
8) Gillis A.S. & Lewis S., (2021) "What is object-oriented programming", Technical Writer and Editor, techtarget 2021
9) Hachadi, Z., (2023) Java Web Development Springboot Security.. (LinkedIn,2023), https://tn.linkedin.com/posts/zakaria-hachadi-176b871b0java − webdevelopment − springboot − activity − 7047332925712785408 − x4Y 0
10) Kumar S., & Sankar P., (2021) Object Oriented Programming (Java). No. 11, Veerabathra Nagar, Part II, 8th Street, Medavakkam, Chennai – 600 100, Tamil Nadu, India.
11) Lakshmanamoorthy R., (2021) "Object-Oriented Programming with Python", Analytics India Magazine.
12) Martinez D., Remegio A., &Lincopinis D., (2023) A Review on Java Programming Language. https://www.researchgate.net/?enrichId=rgreq-075590a898a9eaee62ebf0b14dbdaee2-XXX&enrichSource=Y292ZXJQYWdlOzM3MTE2Njc0NDtBUzoxMTQzMTI4MTE2MzE2MzU3MkAxNjg1NTIxOTc3MzM0&el=1_x_1&_esc=publicationCoverPdf
13) Martinez D., Remegio H., &Lincopinis D., (2023) A Review on Java Programming Language. Western Mindanao State University https://www.researchgate.net/?enrichId=rgreq-075590a898a9eaee62ebf0b14dbdaee2-XXX&enrichSource=Y292ZXJQYWdlOzM3MTE2Njc0NDtBUzoxMTQzMTI4MTE2MzE2MzU3MkAxNjg1NTIxOTc3MzM0&el=1_x_1&_esc=publicationCoverPdf
14) Nagineni R., (2021) A Research on Object Oriented Programming and Its Concepts. esh Babu Nagineni, International Journal of Advanced Trends in Computer Science and Engineering, 10(2), March - April 2021, 746 – 749 https://doi.org/10.30534/ijatcse/2021/401022021
15) Nagineni, R. B. (2021). A research on object-oriented programming and its concepts. International Journal of Advanced Trends in Computer Science and Engineering, 10*(2), 1-10. https://doi.org/10.30534/ijatcse/2021/401022021
16) Nzerue-Kenneth P.E., Onu F.U., Denis A.U., Igwe J.S., Ogbu N.H. (2023), Detailed Study of the Object-Oriented Programming (OOP) Features in Python. British Journal of Computer, Networking and Information Technology 6(1), 83-93. DOI: 10.52589/BJCNITFACSOJAO
17) Palanciuc D., & Pop F., (2021) Implementing Replication of Objects in DOORS—The Object-Oriented Runtime System for Edge Computing. Sensors 2021, 21, 7883. https://doi.org/10.3390/s21237883
18) Parmar M., & Parmar S., (2024) Survey on Concept of Object-Oriented Programming.

International Journal of Scientific Research in Computer Science, Engineering and Information Technology. https://doi.org/10.32628/CSEIT243647

19) Perrelli M., Cosco F., Carbone G., Lenzo B., & Mundo D., (2021) On the Benefits of Using Object-Oriented Programming for the Objective Evaluation of Vehicle Dynamic Performance in Concurrent Simulations. Machines 2021, 9, 41. https://doi.org/10.3390/machines9020041
20) Radha G., & Gaytri G., (2023) Basic Concept Of Object-Oriented Programming (OOP). International Research Journal of Modernization in Engineering Technology and Science. https://www.doi.org/10.56726/IRJMETS31010
21) Rau R., (2020) "Object-oriented programming (OOP)". International Research Journal of Engineering and Technology (IRJET)
22) Raut, R. S. (2020). Research paper on object-oriented programming (OOP). *International Research Journal of Engineering and Technology, 7*(10), 1-8. https://www.irjet.net
23) Sofronas D., Margounakis D., Rigou M., Tambouris E., &Pachidis T., (2023) SQMetrics: An Educational Software Quality Assessment Tool for Java. Knowledge 2023, 3, 557–599. https://doi.org/10.3390/knowledge3040036
24) StudySmarter (2024) Inheritance in OOPs. Inheritance in Oops: Advantages, Types, Importance (studysmarter.co.uk) Accessed June 2024
25) Surya, M., & Padmavathi, S. (2019). A survey of object-oriented programming languages. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 5*(2), 1-12. https://doi.org/10.32628/CSEIT195248
26) Vamanmehtre, V., & Bhalla, A. (2019). Analysis of object-oriented programming. *IRE Journals, 3*(6), 1-6. https://doi.org/10.2456-8880