ISSN NO-2584-2706

Java's Object-Oriented Paradox: Why it's not Purely OOP

Sandhya Dahake; Sahil Sheikh; Wilson Francis
Department of Master in Computer Application, G H Raisoni College of Engineering and Management
Nagpur, Maharashtra, India

Abstract:
In this paper authors trying to studies that Java is not strictly object-oriented, even though it is commonly considered an object-oriented programming (OOP) language. Java contains primitive data types (char, double, int, etc.) that are not part of the object model, but a pure OOP language considers everything as an object. Java also departs from rigid OOP standards by supporting static methods and procedural programming. Features like operator overloading and real multiple inheritance are absent from Java in contrast to completely object-oriented languages like Smalltalk. Because of these drawbacks, Java is not strictly object-oriented but rather categorized as a hybrid language. These features will be thoroughly examined in this paper, along with the reasons why Java does not entirely adhere to the rules of pure object-oriented programming.

Keywords:
Java, Object-Oriented Programming (OOP), Pure Object-Oriented Language, Primitive Data Types, Procedural Programming.

1. **Introduction**

Among the most widely used programming languages, Java is renowned for its resilience, platform freedom, and strict adherence to object-oriented programming (OOP) concepts. Everything in a program should be represented as an object, according to the theory behind object-oriented programming, which supports concepts like abstraction, polymorphism, inheritance, and encapsulation. But in a language that is entirely object-oriented, all entities—including simple data types and functions—must be regarded as objects.

Although Java is promoted as an object-oriented language, it has a number of features that depart from the fundamentalof OOP. Java is not entirely object-oriented, as seen by the usage of static methods, the existence of primitive data types (int, char, double, etc.), and the support for procedural programming features. Furthermore, unlike fully object-oriented languages like Smalltalk or Ruby, Java lacks operator overloading and does not support multiple inheritance through classes.
This study looks at what makes a pure object-oriented language, how Java deviates from these standards, and why Java should be categorized as a hybrid programming language instead of a totally object-oriented one.

2. Background

By grouping code into objects, the programming paradigm known as object-oriented programming (OOP) encourages modularity, reusability, and maintainability. All data types and functions are treated as objects in a totally object-oriented language, which rigorously adheres to the fundamental OOP concepts of abstraction, polymorphism, inheritance, and encapsulation. The fact that languages like Smalltalk and Ruby do not permit non-object elements makes them entirely object-oriented. Java was created as a high-level, platform-independent, and secure programming language and was first released by Sun Microsystems in 1995. It uses procedural programming components for flexibility and speed optimization, even if it adheres to OOP
essential characteristics as a hybrid language that combines procedural and object-oriented programming ideas [5].

In order to prove that Java is not a solely object-oriented programming language, this

DOI: https://doi.org/10.5281/zenodo.15472828

paper examines Java's non-OOP features, contrasts them with those of other languages, and provides evidence.

4. Methodology

The reason Java is not a strictly object-oriented programming language is examined in this research article using a theoretical and comparative analysis technique. The following are the main steps in the methodology:

4.1 Literature Review:

A comprehensive analysis of scholarly works, books, and official Java documentation was done in order to comprehend the fundamentals of Object-Oriented Programming (OOP) [2] and how Java applies them [1], [4].

4.2 DefiningPureObject-Oriented Characteristics:

Characteristics of a purely object-oriented language were specified in order to create a clear evaluation framework. These consist of:

4.2.1 Everything must be an object:

Even basic values like characters and numbers should be objects in a pure OOP language [3]. Nevertheless, Java contains primitive data types that are not objects, like int, double, and char [6]. In java,

```
int x = 10;
```

In contrast, a pure OOP language like Smalltalk treats everything, even numbers, as objects [3].

4.2.2 No primitive data types:

To wrap primitives as objects, Java has wrapper classes like Integer and Double; nevertheless, this is merely a workaround and not a true OOP solution [4].

```
Integer y = 20;
```

Java still permits the direct use of primitive types, though, which goes against the fundamentals of OOP[6].

4.2.3 No static methods or procedural programming constructs:

Java permits static methods to be called without first generating an object, however in a pure OOP language, all methods should be a component of objects [1], [5].

```
class Example {
 static void showMessage(){
System.out.println("This is a static method.");
 }
 }
public class Main {
 public static void main(String[] args) {
 Example.showMessage();
  }
 }
```

This breaks the object-oriented approach since showMessage() is not associated with an instance of Example .

4.2.4 OperatorOverloadinAndMultipleInance:

Java does not support operator overloading, unlike C++. Also, Java does not allow multiple inheritance through classes, only via interfaces [3].

4.3 Comparative Analysis:

To identify differences, Java's characteristics were contrasted with those of strictly object-oriented languages like Smalltalk and Ruby [3]. Java's designation as a hybrid language was aided by this investigation [5].

4.4 Code-Based Evaluation:

The following are some examples that were used to demonstrate Java's non-object-oriented programming features:

- The existence of primitive data types (int, double, etc.) [6].
- The use of static methods and variables [1], [5].
- The ability to write procedural-style code within the main() method [5].
- The absence of operator overloading and multiple inheritance through classes [1], [3].

4.5. Conclusion Based on Findings:

The evidence that was gathered was reviewed to confirm that Java, despite being heavily object-oriented, has aspects that are not object-oriented, which prevents it from being classed as a language that is purely in the object-oriented category [5].

5. Result And Discussion

Java cannot be categorized as entirely object-oriented when its characteristics are compared to the requirements for a purely object-oriented language [3]. Java's design choices contain a number of non-object-oriented components that are more in line with procedural programming, even though the language strongly supports object-oriented programming concepts [5].

5.1 Primitive Data Types:

The pure OOP tenet that everything should be an object is directly violated by Java's support for primitive types like int, char, double, and boolean [6]. Like,

```
int number=5;
```

In contrast, purely object-oriented languages like Smalltalk represent even numbers as objects [3].

5.2 Static Methods and Variables:

Variables and methods can be specified as static in Java, which means they belong to the class and not any instance of an object. The object-oriented tenet that all behavior should belong to objects is broken by this procedural activity [1], [5]. For example:

```
class Utility {  static void displayMessage() {
System.out.println("Static method called");
}
}
```

Here, displayMessage() can be called without creating an object of Utility.

5.3 Procedural Programming Support:

It is possible to write code in a totally procedural way, where no objects are created, using Java's main() function, which acts as the program entry point. Although this flexibility is useful, it goes against the idea of pure OOP [5].
For Example:

```
public class Main { public static void
main(String[] args) {
System.out.println("Hello, world!");
 }
}
```

In contrast to pure OOP languages, where all activities must take place through objects, no objects are needed in order to print a message.

5.4 Lack of Operator Overloading:

Unlike C++ or Smalltalk, Java does not allow operators like +, -, *, etc. to be overloaded to operate with custom objects [1], [3]. This lessens Java's commitment to true OOP and limits the expressiveness of objects [5].

5.5 No True Multiple Inheritance:

As a design tradeoff to prevent ambiguity, Java only permits multiple inheritance through interfaces rather than classes [1]. Although this method is effective, pure OOP systems like Smalltalk permit multiple inheritance directly through classes, therefore it is not a completely object-oriented feature [3].

6. Conclusion

Although Java is frequently referred to as an object-oriented language, a closer look reveals that Java is not entirely object-oriented [5]. Everything, including basic data types and functions, must be handled as an object in a language that is exclusively object-oriented. However, Java provides primitive data types that are not part of the object paradigm, such as int, char, and double [6]. Additionally, it violates the fundamental object-oriented tenet that behavior should belong to objects by permitting static methods to be called without first generating objects [1], [5].
Java also doesn't have operator overloading [1], only allows multiple inheritance for interfaces [3], and supports procedural programming with its main () function [5]. These characteristics, which combine procedural and object-oriented programming techniques, are more in line with hybrid programming.
These design decisions shift Java away from being solely object-oriented even though they improve simplicity and efficiency. In order to meet the demands of real-world programming, Java is best categorized as a hybrid language, combining procedural freedom with object-oriented strength [5].

7. References

[1] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley, The Java Language Specification, Oracle, 2020. [Online]. Available: https://docs.oracle.com/javase/specs/

[2] H. Schildt, Java: The Complete Reference, 11th ed. New York: McGraw-Hill Education, 2018.

[3] Kay, “The Early History of Smalltalk,” ACM SIGPLAN Notices, vol. 28, no. 3, pp. 69–95, Mar. 1993, doi: 10.1145/155360.155364.

[4] Oracle, Java Tutorials: Object-Oriented Programming Concepts, 2024. [Online]. Available: https://docs.oracle.com/javase/tutorial/java/concepts/

[5] Smith, J. (2020). "Analyzing Object-Oriented and Procedural Features in Java." Proceedings of the International Conference on Software Engineering (ICSE).

[6] Miller, R., & Davis, L. (2018). "Primitive Data Types in Java: A Challenge to Object-Oriented Purity." IEEE Symposium on Programming Paradigms.

[7] Stroustrup, B. (2013). The C++ Programming Language, 4th ed. Addison-Wesley.

[8] Bloch, J. (2018). Effective Java, 3rd ed. Addison-Wesley.

[9] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.

[10] Oracle. (2024). "Java and Multiple Inheritance: Design Choices and Limitations." [Online]. Available: https://docs.oracle.com/javase/tutorial/java/IandI/multipleinheritance.html

[11] Lee, K., & Anderson, P. (2019). "Static Methods and Procedural Programming in Java: An OOP Trade-off." Journal of Software Engineering Studies, vol. 15, no. 2, pp. 45-58.

[12] "Why doesn't Java offer operator overloading?" Stack Overflow, 2008. [Online].Available: https://stackoverflow.com/questions/77718/why-doesnt-java-offer-operator-overloading

[13] A. Obregon, "A Guide to Object-Oriented Programming in Java," Medium, 2023.[Online].Available: https://medium.com/@AlexanderObregon/a-guide-to-object-oriented-programming-in-java-89dc4544837f