https://doi.org/10.26682/sjuod.2023.26.2.35

Journal of University of Duhok., Vol. 26, No.2 (Pure and Engineering Sciences), Pp 379-389, 2023 (Special Issue)

COMPARATIVE ANALYSIS OF ENCAPSULATION IN JAVA AND PYTHON: SYNTAX AND IMPLEMENTATION DIFFERENCES

Skala Kamaran Omer*, Star Dawood Mirkhan *, Nyan Najat Hussein*, Aveen Zuber Ali*, Tarik Ahmed Rashid *, and Poornima Nedunchezhian**

*Computer Science and Engineering, University of Kurdistan Hewler, Kurdistan Region- Iraq

**School of Computer Science and Engineering, Vellore Institute of Technology, Vellore- India

(Accepted for Publication: November 27, 2023)

ABSTRACT

Encapsulation is a fundamental principle of object-oriented programming, which allows for the hiding of implementation details and the protection of data from unauthorized access. This paper evaluates the concept of encapsulation in the programming languages Java and Python. The study examines the ways in which encapsulation is implemented in each language, including access modifiers and methods of data hiding. This research paper's findings demonstrate that while both languages support encapsulation, the syntax and specific implementation vary in each of them. In particular, Java's use of access modifiers such as private, protected and public allows for a stricter level of encapsulation compared to Python's use of the '_' and '__' prefix to denote private variables and methods.

***KEYWORDS**:* Encapsulation; Java; Python; Object-oriented programming; Security

1. INTRODUCTION

Encapsulation is one of the fundamentals of OOP which is a protective barrier that prevents the code from gaining access to the data. Encapsulation provides data hiding, increasing the flexibility, reusability, and easy testing. Data encapsulation relies on access modifiers. By utilizing encapsulation, a structured data object values or state are kept inside a class secretly, preventing unauthorized parties from having direct access to them. As it is one of the fundamentals of OOP, all the programming languages that are object-oriented programs provides encapsulation, but the mechanism of implementation might differ, in this paper java and python programing languages will be compered in term of providing encapsulation. In java to implement encapsulation it needs to declare a class as personal and using mechanisms such as public setters /getters to alteration and interpret the values of fluid, whereas in python it differs. Python and java differ in term of simplicity, getter and setter method, the efficiency of debugging and testing and number of standard libraries, but they can match in term of security, readability, both are high level programming languages and many similarities that will be discussed in this paper. Objective of this research is to analyses java and python by mentioning each one's characteristics. The outline of the rest of the paper is as follow: the background for this work is presented in Section 2. In Section 3, we demonstrate the implementation of encapsulation in java and python. The discussion of this paper is given in Section 4; and finally, we concluded

skala.kamaran@ukh.edu.krd
satar.dawood@ukh.edu.krd,
nyan.najat@ukh.edu.krd,
aveen.zuber@ukh.edu.krd
tarik.ahmed@ukh.edu.krd

our work in Section 5.

2. BACKGROUND

2.1 History of Encapsulation in Java and Python

Despite the numerous languages that may be beneficial for programmers, in this paper, two languages are explained and compared for the purpose of simplicity. The two mentioned programming languages, Java and Python, are well-known and highly rated on reputable websites. They are well-liked, and there is a good amount of demand for them in the labor market. A programming language for a beginner needs to have some qualities such as: credibility, simplicity, accessibility, and ease of understanding. The group has looked into the encapsulation concept in two languages using a variety of sources, including books, journals, and research papers. Encapsulation refers to data being wrapped up under a single unit. It is the mechanism that connects the data that the code manipulates with the code itself (Khoirom, Sonia, Laikhuram, Laishram, and Singh, 2020). Encapsulation can also be thought of as a barrier that stops code from outside the barrier from accessing the data. In this, a class's variables or data are hidden from all other classes and may only be accessed through any member functions of the class in which they were declared (Ma and Foster, 2007). The Java programming language was developed by James Gosling, Mike Sheridan, and Patrick Naughton in June 1991 at Sun Microsystems, a company James Gosling founded. It was published as a key product of Sun Microsystems in 1995. Java is the unseen driving factor behind many of the common tools and programs that run on them. Java has the potential to be a great OOPs language (Johnson and Chandran, 2021).

Encapsulation is a term used in object-oriented programming to build abstract datatypes that should only allow external interface modification. Programs that are easier to maintain, troubleshoot and reason about are the result. Declaring variables with various access modifiers, such as private or public, can help you control encapsulation to some extent. However, only the names of the variables used in programming languages with reference semantics, like Java, are protected; the actual objects referenced by the variables are not. Given that references to objects may be returned through method calls, it is conceivable for an object that is external for a compound object to gain references to objects kept in the compound object's private variables (Skoglund, 2003).

Java requires encapsulation because it manages how data is accessible:

- modifies the code according to the requirements
- assists in achieving a loose couple
- simplifies our application to achieve the goal

Additionally, it enables you to alter the portion of the code without impairing any other program features or lines of code. Python was created by Guido van Rossum at Central Wiskunde & Informatica (CWI) in the Netherlands in the late 1980s and implemented in December 1989. A replacement for the ABC language, which is capable of handling exceptions and interacting with the operating system Amoeba, has been suggested Python. Python might be a good choice for both learning and global programming. Python is a powerful high-level, object-oriented programming language. In recent years, Python has earned a reputation as an extremely user- and beginner-friendly language (Johnson and Chandran, 2021) We need Encapsulation in Python therefore Encapsulation produces understandable, well-defined code.

The main benefit of implementing encapsulation in Python is that as users, we are not required to understand the method and data design and can instead concentrate on using these useful, encapsulated units for our applications. As a result, the code becomes cleaner and more organized. Additionally, the user experience is substantially enhanced, which makes it simpler to

comprehend apps as a whole. Encapsulation protects against Accidental Modification or Removal. Encapsulation also has the benefit of preventing the unintentional alteration of data and techniques. In Python, encapsulation is accomplished using access modifiers. These access modifiers offer a fantastic user experience in terms of security by ensuring that access conditions are not violated.

2.2 Previous Related Works

In the object-oriented literature, the use of encapsulation and information hiding is emphasized since these techniques are thought to speed up software development, enhance its quality, and make maintenance easier. For instance, the Java Programming Language Code Conventions state the following (Cramer, Friedman, Miller, Seberger, Wilson, and Wolczko, 1997): "Don't make any instance or class variable public without a good reason". If programmers have the discipline to follow such guidelines and coding standards that advise them to code in specific ways, it may improve the quality of the code created. Additionally, some methods and tools support the ideas of information hiding and encapsulation. For instance, member variables in Smalltalk are by design inaccessible to users outside of their classes. As a result, getter and setter methods must be created by the programmer in order to allow outside access to the objects that member variables refer to. This should lessen the possibility of a programmer accidentally breaking encapsulation. Tools for automatically creating code may also assist programmers by creating code for member variables that are declared with restricted access, such as private, and only creating getter and setter methods for such variables when necessary. As a result, there should be less chance that variables may accidentally become more accessible than necessary. By pointing out member variables that have been defined as public, for instance, static analysis tools can be used to analyze programs and show programmers where encapsulation is weak. However, the programmer need not always employ the established tools, methods, and techniques for encapsulation, information hiding, and representation exposing. For instance, a member variable in Java must be expressly declared as private by the programmer. Additionally, the actual evidence suggests that programmers may not always enforce information hiding and encapsulation.

Menzies and Haynes (1996) examined the degree of information hiding and encapsulation in Smalltalk applications by analyzing Smalltalk classes. They reasoned that there should be few calls to other classes if the program follows the notion of encapsulation. The research of 2000 classes across 5 applications, however, reveals a low prevalence of information concealment. Using static analysis techniques, Elish and Offutt (2002) looked at 100 open-source Java classes and their compatibility with 16 different coding standards. According to their research results, member variables shouldn't be made public is the third most problematic coding technique. In 15% of the classes inspected, this practice is disallowed. 28 students studying object-oriented programming were questioned by Fleury about how well they understood certain object-oriented principles (Fleury, 2001). Many students in this survey believed that decreasing the number of classes and lines of code was more crucial than encapsulation. For example, some students thought that a class with accessor methods and all member variables set to the private is more preferable than a class with public member variables and no accessor methods because the latter had a larger code. There could be several justifications for not employing strategies for regulating encapsulation and information concealment. One explanation might be that the programming languages don't provide enough support. For instance, Evered and Menger contend that since C++ does not permit separating the declaration of the public member variables from the private member variables, students have

difficulty employing encapsulation and information hiding in the language (Eustace and Hay, 2000). In the final program the amount of encapsulation may be negatively impacted by other strategies that were applied. For instance, as Snyder pointed out (1986), the inheritance method in Smalltalk allows the subclass full access to the member variables of the superclass, which breaks encapsulation and makes it more difficult to update the superclass without impacting the clients. The difficulty of teaching object orientation may be another reason to abandon the current methods, which means that individuals who study it do not fully understand the ideas of encapsulation and information hiding. However, Klling believes that the fundamental cause of this might be the lack of clear, comprehensible, and consistent notions in the programming languages used to teach object orientation (Kölling, 1999).

Although empirical evidence suggests that encapsulation and information hiding are not fully utilized in object-oriented programming, there is insufficient information to determine whether this has any effect on the caliber of the software created. Additionally, there are few empirical studies examining the impact of representation exposure on software quality. several structural interviews were performed by Nilsen (2007) with software engineers in the software industry to learn more about these topics. Table 1 explains the subjects' familiarity with different programming languages.

Table 1. Subjects' familiarity with different programming languages. (Nilsen, 2007).

Subj.	Java	C++	Python	Delphi
A				✓
B				✓
C	✓		✓	
D	✓	✓		
E	✓	✓		✓
F		✓		
G	✓	✓	✓	✓
H		✓		

The subjects that are set as the default access modifiers of private or protected when declaring member variables. Subject A modifies the variable's declaration from private to protected when Subject A foresees that the class containing the variable will later be subclassed and that the subclass will require the variable. Member variables that Subjects C and E initiated are by default protected, subject D did so because of the organization's culture, and subject E did so facilitate inheritance beforehand. Subjects D, F, and G use private by default unless they intend to create subclasses that will require the variable, in which case they will declare it directly with the protected modifier.

In the initial stages of development, Subjects C, E, and F stated that they rarely declare variables as public in order to conduct some testing; however, once the testing is completed, they always change the declaration. Subjects A and B raised the issue of changing a private declared variable to the public rather than reworking the design when it needs to be accessed from an external object when there is a time constraint. "There is not always the time to do things exactly the way you want", says subject B. A Public variable may be present in a class that is completely internal to a component and still not

accessible from the outside according to subject H. When a class just acts as a container for data variables and there is no implementation inside the class, that is where subject C and H are allowed to declare variables as public. Subjects H, C, and G stated that they have never replaced public variables for private or protected variables for performance reasons. For performance reasons, subject F had once created a graphical package and used public variables.

Python is an eminently flexible programming language. Aim of developing Java was to develop a secured platform. Each and every application has a Security Manager. The use of OOP in Java facilitates the development of standardized programs and code that can be reused. Python has a low learning curve since it is straightforward, easy to use, and a productive language. Table 2 explains some common features between Java and Python.

Table(2):- Some common features of Java and Python.

Features	Java	Python
High-level language	✓	✓
Expressive		✓
Easy		✓
Free and Open Source	✓	✓
Object-Oriented	✓	✓
Secured	✓	
Platform Independent	✓	
Robust	✓	
Multi-thread	✓	
Portable	✓	✓
Distributed	✓	
Expressive		✓
Large-Standard Library		✓
Dynamically Typed		✓
Interpreted Language		✓

Table(3):-Features of Encapsulation in each language

Encapsulation	**Java**	**Python**
Flexible Programs	✓	✓
Easy debugging & testing	✓	
Reusability	✓	✓
Hiding data	✓	
Getter & Setter method	✓	
Total control	✓	✓
Security	✓	✓
Simplicity		✓
Aesthetics	✓	✓
Readability	✓	✓

Coding becomes easier to read and maintain when data and methods are combined into a class. It uses private and protected access levels to secure an object from illegal access, and increases security by protecting the code and logic from outside inheritance. A class's defined private members cannot be used by other classes because of its private members. Getters and setters differ from one another in Python from those in other object-oriented programming languages. In general, getters and setters are used in object-oriented programming to maintain data encapsulation. Every program benefits from having well-defined interactions thanks to encapsulation. DRY guarantees data security and prevents unintentional data access. Any modifications to one section of the code won't affect the other. The programs can be safely updated. It encourages a simple user experience without revealing any back-end difficulties. The reusability of Python code is a key component of the object-oriented approach. It makes the code easier to read.

In Python encapsulation, the data is kept secret outside of the object declaration. As the code is extremely secure and cannot be read by other sources, this helps protect data from breaches. It also facilitates the creation of user-friendly experiences by developers. Coding becomes easily readable and maintained when data and methods are combined into a class. An object is shielded from unwanted access through encapsulation. The user is unaware of what these setter and getter functions are doing. To avoid unintentional data change, private and secured access levels are available. Data concealment is when the user is not aware of what was happening in the background.

3. Implementation of Encapsulation in Java and Python

Consider the usage of Laptop, sure most of us enjoying while using the laptop for daily works or general usages, we know how to use it and what is the usage of monitor, mouse, keyboard, speaker, but not most of us need to know how the laptop has been build. We are unaware of the method used to obtain the components or the length of time required to design the model. To us, none of these factors matter; the only thing that counts is the final result. Essentially, this is how encapsulation operates. It means that encapsulation in programing and relative to user is like users do not have to understand how classes are implemented or saved. Users will only be aware of the values being submitted and initialized. The following are some reasons why capsulation is used:

• Improved control over class characteristics and methods and gives complete control over the class's data members and data methods.

• Can make class attributes read-only by using only get method, and can make them write-only by using only set method.
• Improved data security.
• Decreased programming errors.
• Simplifies application maintenance and improves application comprehension.
• Quicker and easier application debugging.

3.1 Comparison of the Design Structure of each Language

In Java, encapsulating the variables of a class means only the methods of that class can access them. Other classes cannot access them. The variables of a class are hidden from other classes during encapsulation, and only the methods of the class whereby they are located can better access them. The data members and methods of a class are declared as either private or protected is one way to achieve encapsulation. However, Python lacks direct access modifiers like protected, private, and public. Using single and double underscores, we can accomplish this. Access modifiers restrict access to a class's methods and variables.

3.2 Advantages and Disadvantages of Encapsulation in Java and Python

3.2.1 Advantages of Encapsulation in Java

• One of the advantages of encapsulation in Java is that users would not be having any idea about the implementation of the classes. The user would be having information about how values are passed and initialized.
• Another advantage of encapsulation in Java is that Data abstraction can be achieved easily. By this, one can hide the operation and structure of the actual program from the user and can present only the required information. Having said that, all the data and the code are saved from any external interference.
• The encapsulated code is very easy to debug and quite easy to test.
• Encapsulation makes it simple to reuse code, modify procedures, and implement new program requirements.

3.2.2 Disadvantages of Encapsulation in Java

• The disadvantage of encapsulation in Java is that we are supposed to provide every method with the specifier and in that case, the length of the code increases.
• The user is supposed to provide additional information for every method if the size of the code increases.
• Encapsulation increases the duration of execution of the program. The reason behind an increase in the duration of the program is the addition of instructions.
• Complex the procedure.

3.2.3 Advantages of Encapsulation in Python

• Encapsulation in Python provides protection to objects from unauthorized access.
• Another advantage is that all data can be combined into one single unit.
• It also offers help in preventing accidental data modification.

3.2.4 Disadvantages of Encapsulation in Python

• Encapsulation in Python may occasionally necessitate additional coding by the programmer.
• Encapsulation prevents the linkage between the visible and invisible data, which speeds up the objects'operation.
• Private information cannot be accessed outside of class.
• For the purpose of encapsulation in Python, there is a need of writing lengthy codes by the programmer.

3.3 Implementation of Python

As we know encapsulation is wrapping up all data in one single unit. In Java, encapsulation is a process, and in that process, all data or variables and codes are integrated as one single unit. Through this, other classes cannot have access to them, but only the methods of the class can have access. Encapsulation can be achieved by labeling all the variables as private in the class, and by writing the public methods in the class which means if we have public get and set methods, we can access them by setter method put value and then by getter method get the value. In the case of

Python, by limiting permissions, Python codes encapsulate and conceal backend implementation details; users do not have direct access to modifiers including private, public, and protected. Therefore, in Python, it can be achieved by the use of single and double underscores. The access given to the variables are as follows:

Public: Access to the data is available to all objects.

Protect: Access is restricted to members of the same class or their descendants.

private: Only members of the same class have access.

Table(4):- Accessibility of each of the state of encapsulation.

Type	Use your own class to access	Use sub class to access	Use objects to access
Private	Yes	No	No
Protect	Yes	Yes	No
Public	Yes	Yes	Yes

Table (5):- Defining syntax of encapsulation in Python and Java

Type	Python	Java
Private	___VaribleName	private String VaribleName
Protect	_VaribleNam	protect String VaribleName
Public	VaribleName	public String VaribleName

```
class item:
    def __init__(self,name,base_price,sell_price):
        self.name=name #Public type
        self._base_price = base_price # Protect type
        self.__sell_price=sell_price #Private Type

    def get_value(self):
        return {"Name":self.name,"Base price":self._base_price,"Sell Price":self.__sell_price}

obj=item("BMW",20000,35000) # setter , setting the value to the varibles
print(obj.get_value()) # (getter) this function will get all the value we can print or use it
print(obj.name) #this will prine (BMW) means just (name) that is public varible from the Item class
print(obj._base_price) #this will rasi an error (AttributeError: 'item' object has no attribute 'base_price')
print(obj.__sell_price) #this will rasi an error (AttributeError: 'item' object has no attribute '__sell_price')
```

Fig. (1):- Python Implementation.

3.4 Implementation of Java

To declare the variables of a class to be private and to inspect and change the values of the variables, provide public setter and getter methods.

```java
class Item{
    public String name;
    protected int base_price;
    private int sell_price;
    // Getter
  public int g
    return bas
  }
  // Setter
  public void
    this.base_
  }

   // Getter
  public int getSell_price() {
    return sell_price;
  }
  // Setter
  public void setSell_price(int put_sell_price) {
    this.sell_price = put_sell_price;
  }

}
public static void main(String[] args) {
    // TODO code application logic here
    Item obj=new Item();
    obj.name="BMW";
    obj.setBase_price(20000);
    obj.setSell_price(35000);
    System.out.println("Item name :"+obj.name+", base Price :"
            +obj.getBase_price()+", Sell Price : "+obj.getSell_price());
    obj.sell_price=20000;// this will cause error because of private type

}
```

Fig. (2):- Getters and Setters in Java

3.5 The difference between Setter and Getter (Python vs. Java)

In Object Oriented Programming, getters and setters are used because of encapsulation. We maintain the privacy of our variables to make sure that only getters and setters may access and modify them. In java we should create two methods with different name to implement setters and getters but in python we can put the same name for getters and setters but with different attributes and properties.

```java
// Getter
public int getSell_price() {
  return sell_price;
}
// Setter
public void setSell_price(int put_sell_price) {
  this.sell_price = put_sell_price;
}
```

Fig. (3):- Java example

```
# Getter
@property
def sell_price(self):
    return self.__sell_price
@sell_price.setter
def sell_price(self,sell_price):
    self.__sell_price=sell_price
```

Fig. (4):- Python example.

4. DISCUSSION

Encapsulation is one of the basics of object-oriented programming. It describes the integration of data and the operations that utilize that data. By using encapsulation, a structured data object's values or state are concealed inside a class, preventing unauthorized parties from having direct access to them. Data is encapsulated when it is combined into a single unit. It is a protective barrier that forbids unauthorized access to data from the outside. Data encapsulation depends heavily on access modifiers. Public, protected, and private members are available in Python. Java has a default, protected, private, public, and private members. Encapsulation is a Java technique for data hiding, preventing other classes from accessing data stored in private data members. In Java, we can make a class read-only or write-only by including only the getter and setter methods. Because the languages serve different purposes, Java and Python have different philosophies. Oracle designed Java to be used in the development of complex enterprise software systems. A single man on vacation created Python because he needed a quick way to create useful tools for scientific computing. Despite being fully object-oriented from the beginning, Python was never meant to take the place of Java or the C languages in enterprise software development.

5. CONCLUSION

In conclusion, this study has shown that the concept of encapsulation is present in both Java and Python, but the level of strictness and method of implementation may vary. Java's use of access modifiers such as private, protected and public allows for a stricter level of encapsulation compared to Python's use of the '_' and '__' prefix to denote private variables and methods. This can have implications on the design and development of object-oriented programs, as the level of encapsulation can affect the security and maintainability of the code. The study highlights the importance of understanding the fine distinction of encapsulation in different programming languages and how they can be applied in the design of software systems. It is suggested that further research can be conducted to compare encapsulation in other programming languages, and to investigate the impact of encapsulation on software quality attributes such as security and maintainability.

REFERENCES

Khoirom, S., Sonia, M., Laikhuram, B., Laishram, J., & Singh, T. D. (2020). Comparative analysis of Python and Java for beginners. *Int. Res. J. Eng. Technol*, *7*(8), 4384-4407.

Ma, K. K., & Foster, J. S. (2007). Inferring aliasing and encapsulation properties for Java. *ACM SIGPLAN Notices*, *42*(10), 423-440.

Johnson, B., & Chandran, A. S. (2021). 'Comparison between Python, Java and R progrmming language in machine learning. *Int. Res. J. Modernization Eng. Technol. Sci*, *3*(6), 1-6.

Skoglund, M. (2003). Practical Use of Encapsulation

in Object-Oriented Programming. In *Software Engineering Research and Practice* (pp. 554-560).

Cramer, T., Friedman, R., Miller, T., Seberger, D., Wilson, R., & Wolczko, M. (1997). Compiling Java just in time. *Ieee micro*, *17*(3), 36-43.

Menzies, T., & Haynes, P. (1996). *Empirical observations of class-level encapsulation and inheritance*. Technical report, Department of Software Development, Monash University. (7)

Elish, M. O., & Offutt, J. (2002). The adherence of open source Java programmers to standard coding practices.

Fleury, A. E. (2001). Encapsualtion and reuse as viewed by java students. *ACM SIGCSE Bulletin*, *33*(1), 189-193.

Eustace, K., & Hay, L. (2000, December). A community and knowledge building model in computer education. In *Proceedings of the Australasian conference on Computing education* (pp. 95-102).

Snyder, A. (1986, June). Encapsulation and inheritance in object-oriented programming languages. In *Conference proceedings on Object-oriented programming systems, languages and applications* (pp. 38-45).

Kölling, M. (1999). The problem of teaching object-oriented programming, Part 1: Languages. *Journal of Object-oriented programming*, *11*(8), 8-15. (12)

Nilsen, K. (2007, September). Improving abstraction, encapsulation, and performance within mixed-mode real-time Java applications. In *Proceedings of the 5th international workshop on Java technologies for real-time and embedded systems* (pp. 13-22).

Kniesel, G., & Theisen, D. (2001). JAC—access right based encapsulation for Java. *Software: Practice and Experience*, *31*(6), 555-576.