ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

A Conceptual and Pedagogical Study of Object-Oriented Programming Principles Using Java

Mr. Anand Kumar[1], Dr. Bharathi Ravishankar[2]

[1]Assistant Professor International School of Management Excellence (ISME) Bangalore, India

[2]Associate Professor International School of Management Excellence (ISME) Bangalore, India

DOI: https://doi.org/10.51583/IJLTEMAS.2026.150500033

Received: 02 May 2026; Accepted: 07 May 2026; Published: 26 May 2026

ABSTRACT

The concept of Object-Oriented Programming is the foundation of the modern software engineering development and to implement this Java as a programming language is being widely used in the industry. Seeking its importance for real world it has been seen that beginners often struggle to understand the OOPs concept because of their early focus on syntax rather than design thinking.

This article mainly focuses on the core four OOPS principles along with Class and objects. The four pillars are classified into inheritance, polymorphism, encapsulation and abstraction. The study shows how these concepts are used and finally implemented to design structured and maintainable software systems. For reusability of any feature we implement inheritance and for designing the application with the help of interface, we implement abstraction. The paper aims to bridge the gap between theoretical understanding and practical application by explaining OOP concepts using simple explanations and real-world analogies.

INTRODUCTION

When students begin programming in the early semesters of BCA, the focus is usually on writing small programs that work correctly. Syntax, loops, and conditional statements receive most of the attention. However, as programs become longer and more complex, students often struggle to organize their code. This is where Object-Oriented Programming, commonly known as OOPS, becomes essential.

Basically, Object oriented programming is a concept. As a developer ,People make use of the OOPS Concept Principles to design the application by implementing it using Programming language like Java, Python etc.

In the second semester of BCA, OOPS in Java is introduced to help students move beyond basic coding and start thinking about software structure. At this stage, learners are not expected to build large applications, but they are encouraged to understand how real-world software is organized. OOPS provides a systematic way to think about programs by relating them to real-world objects and interactions.

Object-Oriented Programming is not just a programming technique but a way of thinking. Instead of focusing only on functions and instructions, OOPS encourages programmers to think in terms of objects that contain both data and behavior. This approach makes programs easier to understand, modify, and extend over time.

Java is particularly suitable for teaching OOPS because it is designed around object-oriented principles. Almost everything in Java revolves around classes and objects. This makes Java an ideal language for beginners to learn structured and organized programming.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

LITERATURE REVIEW

Existing literature on Object-Oriented Programming primarily focuses on its role in developing scalable and maintainable software systems, with significant emphasis on advanced topics such as design patterns, frameworks, and performance optimization. Standard textbooks and professional references provide detailed technical explanations of OOP principles but often assume prior programming maturity.

Several studies have reported that beginners face significant challenges when transitioning from procedural programming to object-oriented thinking. Learners tend to focus on memorizing syntax without fully understanding object interactions, inheritance hierarchies, or abstraction mechanisms. Although Java is one of the most widely used languages for teaching OOP, limited research adopts a conceptual and pedagogical perspective aimed specifically at novice learners.

Recent studies in computer science education highlight the importance of visual aids, real-world examples, and design-oriented teaching strategies for improving conceptual clarity. However, there remains a noticeable gap in literature that integrates simplified explanations, practical case studies, and design diagrams to support beginner-level OOP education. This study attempts to address this gap by presenting an approach that emphasizes conceptual understanding, real-world relevance, and design thinking.

Class and Objects

Classes and objects form the foundation of object-oriented programming. A class can be understood as a blueprint, while an object is an actual instance created from that blueprint. This distinction helps students model real-world entities such as students, accounts, or employees in their programs.

In summary Class is a factory which produces object. We can call object as an instance of class. In java whenever we use the new keyword an object is created in the heap memory of RAM. Basically New Keyword sends request to the class to create object.

Once object is created new keyword will get object address and stores that in reference variable.

Example: A a1 = new A ( ); Here A=Class
A1=Reference Variable Example:

```
class Student {

   int id;

   String name;

}

public class Main {

   public static void main(String[] args) { Student s1 = new
      Student();

      s1.id = 101; s1.name = "Anand";

   }

}
```

The above example demonstrates how a class acts as a template and how objects are created and accessed using reference variables.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Four Pillars of Object Oriented Programming System(OOPS)

1) Inheritance

Inheritance allows one class to reuse the properties and methods of another class. This reduces code duplication and promotes reusability. Students learn how complex systems can be built by extending existing components rather than writing everything from scratch.

In inheritance, a child class reuses the members (fields and methods) of a parent class. Java does not support multiple inheritance with classes, but it does support multiple inheritance through interfaces. An interface defines method signatures without implementations.

Only non-static members of the class get inherited. Static members of the class are never inherited.

Examples 1:

The above is the Parent Class .

```
System.Out.Println ( b1.x); }
                           }
```

The above is the Child Class.

One of the main advantages of inheritance is reusability.

Example 2:

```
class Account {
   int balance = 5000;
   void displayBalance() { System.out.println(balance);
   }
}
class SavingsAccount extends Account { public static void
   main(String[] args) {
      SavingsAccount sa = new SavingsAccount(); sa.displayBalance();
   }
}
```

This example shows how the child class reuses functionality from the parent class without rewriting code.

2) Polymorphism

Polymorphism allows the same method name to behave differently based on context. Through method overloading and method overriding, students understand how flexibility can be built into software systems. This concept prepares learners for advanced programming scenarios.

Here we develop a feature such that it takes more than one form depending on the situation. It is applicable only on methods(not variables).

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Polymorphism can be achieved in 2 ways:

A) **Overriding**

This is also known as Run Time Polymorphism. Here we inherit the method and then we modify the logic of inherited method by once again creating a method in child class.

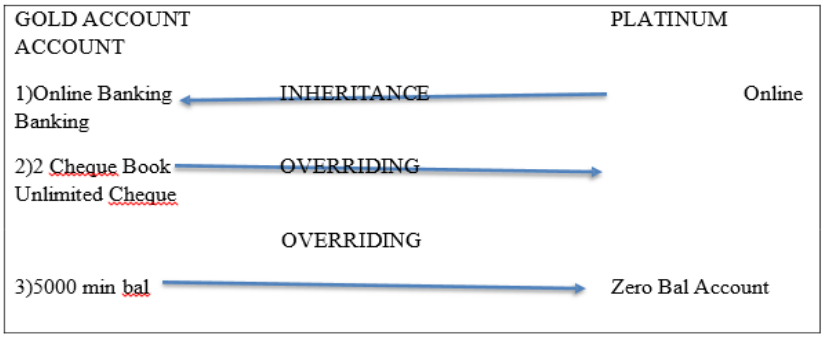
@Override annotation will check whether overriding is happening or not. If overriding is not happening then you will get an error as shown below

```
Public class A
{
Public void test( )
{
System.Out.Println( 100 );
}
}
```

The above code belongs to the Parent Class.

```
Public class B extends A
{
    @override
    Public void tests( ) //ERROR
    System.Out.Println(500);
}
    Public static void main ( )
    {
        B b1 =new B ( ) ;
        B1.test( ) ;
}
}
```

The main advantage of overriding is we can modify the logic of inherited method. Practical example:



B) **Overloading**

This is also known as compile time polymorphism. Here we create a method with a same name in same class more than once provided they have different number of arguments /different types of arguments.

Here multiple methods are created with same name and these multiple methods will be

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

differentiated based on number of arguments or types of arguments.

3)Encapsulation

Encapsulation refers to bundling data and methods together and controlling access to them. Using access specifiers such as private, public, and protected, programmers decide which data should be accessible. This concept teaches students the importance of data protection and controlled access.

This is also known as Data Hiding. Wrapping of data with the methods such that method operate on that data is called encapsulation. Here direct access to the variable is avoided by making the variable private.

It publicly defines setters and getters methods which are used to operate on these variables.

Example:

```
class Account {private int balance;
   public void setBalance(int b) { balance = b;
   }
   public int getBalance() { return balance;
   }
}
```

This approach prevents unauthorized access and maintains data integrity.

Abstraction

Abstraction focuses on hiding unnecessary details and exposing only essential features. In Java, abstraction is achieved using abstract classes and interfaces. This helps students concentrate on what a program does instead of how it does it internally.

Hiding of implementation details is called as Abstraction. We can achieve abstraction in Java using interfaces and abstract classes.

In conclusion, Object-Oriented Programming in Java is more than an academic subject. It teaches students how to think like software developers. Understanding OOPS early in the BCA program helps learners approach complex problems with confidence and clarity.

For many students, OOPS in Java becomes the subject that changes their perception of programming. It shifts the focus from writing code to designing systems, which is a crucial step in becoming a skilled programmer.

Practical Application and Case Study

To enhance real-world relevance, this study incorporates a simple banking system case study. The system demonstrates how OOP principles work together in application development:

- Encapsulation secures account data
- Inheritance supports multiple account types
- Polymorphism enables flexible service behavior

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

Abstraction defines common banking operations

Encapsulation: Account balance is kept private and accessed only through getter and setter methods.

Inheritance: Different account types such as Savings and Current accounts inherit common features from a base Account class.

Polymorphism: Interest calculation methods behave differently for different account types.

Abstraction: Interfaces define essential banking operations without exposing implementation details.

Such examples help learners relate theoretical concepts to practical software design scenarios.

Use of UML and Design Diagrams

To strengthen design thinking, UML class diagrams and inheritance hierarchies are incorporated. These diagrams visually represent relationships between classes and interfaces, enabling students to understand system architecture before coding. Visual representations significantly improve comprehension, especially for beginners.

| Account |

| - accountNumber : int |

| - balance : double |

| + deposit(amount) : void |

| + withdraw(amount) : void |

| + getBalance() : double |

^

|

| |

| SavingsAccount | | CurrentAccount |

| - interestRate | | - overdraft |

| + calcInterest | | + withdraw() |

The above UML class diagram shows a simple banking system developed using Object-Oriented Programming concepts. The Account class is the main class that contains common details such as account number and balance, along with basic operations like deposit and withdrawal. The SavingsAccount and CurrentAccount classes are derived from the Account class. The SavingsAccount class adds functionality for calculating interest, while the CurrentAccount class provides an overdraft facility. This diagram helps in understanding how inheritance, encapsulation, and polymorphism are applied in a real-world banking system.

CONCLUSION

Object Oriented Programming remains a set of ideas in modern software development for designing scalable and maintainable software system. This paper gives a detail conceptual understanding about the

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

core of OOPs principles like class, objects, inheritance

,polymorphism, encapsulation and abstraction. In order to implement this Java as a primary language has been used. The focus was on improving the concepts and fundamentals among the beginners level students rather than focusing on tool specific details.

It has been observed that many learners has found difficulty in understanding the implementation of these concepts due to emphasis on coding mechanics instead of software design principles.

This paper address common learning hurdles faced by the new learners/beginners and the simplified approach and description of the concept in this paper help the students to overcome this challenges of understanding the concept.

Overall, The study suggest that strong foundation in OOPs is a vital part for developing software system in real world scenario. This will not only prepare the students in theoretical aspect but also the students will be in a much better position to apply in advanced software engineering and real world application development.

Research Gap

It has been noticed that in most of the existing literatures the OOPs concepts is being explained in a highly technical manner and there is very limited research that has a focus on the conceptual clarity and fundamentals of these concept with respect to implementation using a a programming language for beginner students.

Also many studies explains the object oriented programming principles but they are not able to relate to how to actually use the concept in software design thinking and implementing in solving real world problem solution.

This creates a gap between understanding the OOPs as a programming requirement. It has been seen that learners understand how to write code using OOPs concept but they fail to apply these principles while designing the complete application. This indicates a clear gap in the literature that addresses Object-Oriented Programming from a conceptual and design-oriented learning perspective.

Future Work

We can incorporate case studies and small enterprise level projects for implementing the Object Oriented Programming principles. These will help to build the learning in the practical aspect and give much more clarity in understanding the concept from broader perspective. Also adding the low level design like UML diagram and following structured design pattern for developing application can help the students to strengthen the concept in more precise manner.

REFERENCES

1. Deitel, H. M., & Deitel, P. J. Java: How to Program. Pearson Education.
2. Bloch, J. Effective Java. Addison-Wesley.
3. Oracle. Java Documentation. Oracle Corporation.
4. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
5. Schildt, H. Java: The Complete Reference. McGraw-Hill Education.
6. Meyer, B. Object-Oriented Software Construction. Prentice Hall.
7. Booch, G. Object-Oriented Analysis and Design with Applications. Addison-Wesley.
8. Larman, C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design. Pearson Education.
9. Stroustrup, B. The C++ Programming Language. Addison-Wesley.

ISSN 2278-2540 | DOI: 10.51583/IJLTEMAS | Volume XV, Issue V, May 2026

(Useful for conceptual comparison of OOP across languages)
10. Oracle Corporation. The Java Language Specification. Oracle Documentation.