International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 9 Issue 1, Jan-Feb 2025 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

A Study on Inheritance and its Type in Object Oriented Programming using JAVA

Sh. Pritosh Godara

Lecturer in Computer Engineering, Government Polytechnic, Uttawar (Palwal), Haryana, India

ABSTRACT

This paper presents the first comprehensive empirical investigation of the widespread use of inheritance in a modern OO programming language. We provide a set of standardized metrics for quantifying inheritance in Java programs. This article will cover the fundamentals of object-oriented programming in Java. Encapsulation, abstraction, inheritance, and polymorphism are among the core principles of this programming language. In this paper, we will discuss. A Study of Inheritance and Its Type in Object-Oriented Programming with Java.

KEYWORDS: *Inheritance, Object Oriented Programming, Java, Language, Encapsulation, Abstraction, Inheritance and Polymorphism, Class, Single Inheritance, Multi-Level Inheritance, Hierarchical Inheritance, Hybrid Inheritance, Subclass*

How to cite this paper: Sh. Pritosh Godara "A Study on Inheritance and its Type in Object Oriented Programming using JAVA" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-9 | Issue-1, February 2025, pp.685-690, URL: www.ijtsrd.com/papers/ijtsrd75049.pdf

IJTSRD75049

INTRODUCTION

Java: Java is both a programming language and a platform. Java is a powerful, secure, and object-oriented programming language.

A platform is any hardware or software environment that allows a program to run. Platform refers to Java's runtime environment (JRE) and API.

In Java, inheritance is a key component of OOP. It is the mechanism in Java that allows one class to inherit features (fields and methods) from another.

In Java, inheritance means generating new classes from existing ones. A class that inherits from another class may reuse its methods and fields. In addition, you can modify your existing class by adding additional fields and methods. [1]

Since the introduction of the object-oriented paradigm, much has been written about the concept of "inheritance". Some people associate the term "object-oriented Ness" with inheritance. Inheritance appears to be a major theme in conversations about good design.

All design patterns include it, frameworks rely on it, and it is even used in UML. Some presentations of the object-oriented paradigm (for example, in textbooks) lay so much emphasis on inheritance that any design without "lots of inheritance" is not good (or certainly not "object-oriented"). At the same time, there is a lot of advice advocating prudence when it comes to inheritance, such as "Favor object composition over class inheritance". There have also been studies that provide inconsistent answers as to its benefits, while also implying that "too much" inheritance is harmful. [2]

Used in Java Inheritance

Class: A class is a collection of objects that share common characteristics/behavior and properties/attributes. Class is not a real-world entity. It is simply a template, blueprint, or prototype from which items are built.

Super Class/Parent Class: The class whose features are inherited is known as a superclass (or a base class or a parent class).

Sub Class/Child Class: A subclass is a class that inherits from another class. In addition to the fields

and methods provided by the superclass, the subclass may include its own.

Reusability: Inheritance supports the concept of "reusability," which means that if we want to construct a new class and there is already a class that contains some of the code we need, we can derive our new class from the current class. By doing so, we reuse the current class's fields and functions.

Object-Oriented Programming in Java:

Object-oriented programming (OOP) is a high-level computer programming language that uses objects and associated procedures inside a programming context to develop software programs. The object-oriented language employs an object-oriented programming technique that ties related data and functions into an object, encouraging reuse of these objects within the same and other systems.

Java is a class-based object-oriented programming (OOP) language centered on the concept of objects. OOP concepts aim to increase code readability and reusability by specifying how to structure a Java program efficiently.

Object-oriented programming combines data and behavior (methods) into a single location (object), making it easier to comprehend how a program operates. [3]

Simula is recognized as the first object-oriented programming language. A true object-oriented programming language is one that represents everything as an object. Smalltalk is regarded as the first genuinely object-oriented programming language.

Object-oriented programming (OOPs) is an approach for simplifying software development and maintenance by establishing basic rules.

Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation [4]

The Benefits of the Object-Oriented Programming Approach

Whether or not you develop programs in an object-oriented manner, you must first create a model of what the software must be capable of and how it should function. Abstraction, encapsulation, inheritance, and polymorphism are fundamental concepts in object-oriented modeling. The general supporters of the object-oriented approach believe that this model provides:

- Better abstractions (modelling information and behavior together)
- Better maintainability (more comprehensible, less fragile)
- Better reusability (classes as encapsulated components) [5]

Review of Literature:

James Gosling invented the Java programming language in 1995. Java has gained popularity as a class-based, object-oriented, and high-level programming language. It is intended to enable the "write once, run anywhere" (WORA) capability, which allows written Java code to execute without further compilation on any platform that supports Java. Java is well-known for its simplicity, portability, and platform independence, making it a popular choice for creating a variety of applications such as mobile apps, web apps, desktop apps, and games. One of Java's distinguishing advantages is its platform independence, as it is compiled into bytecode that may run on any platform that supports the Java Virtual Machine. This allows developers to create code once and run it across multiple platforms without worrying about platform-specific details [Hachadi,2023]. Furthermore, Java is widely utilized in industry and academia, with numerous firms relying on it for important business applications. [6]

The most frequently mentioned inheritance-related metrics are Chidamber and Kemerer's DIT and NOC measures [7]. A class's DIT is defined as the length of the longest path from the class to the root of its inheritance hierarchy. The authors claim that the deeper the class, the more complex it will be because it will inherit from more ancestors, but also the greater the opportunity for reuse. NOC for a class is defined as the number of its immediate subclasses. The authors proposed that having more children leads to more reuse, but it also increases the chance of erroneous abstraction. They also discovered that NOC provides an indication of the influence a class has on the design.

Daly et al. investigated the impact of depth of inheritance on maintenance, as assessed by the time required to complete a maintenance activity [8]. Their findings indicated that inheritance had a detrimental

impact on maintenance time. This study was later duplicated, and the results indicated the opposite effect: inheritance increased maintenance time. The fact that these two experiments produced such disparate results shows that there may be more to heredity than just "depth," yet both were modest enough that an influence other than inheritance could have been noticed.

Objectives:

- To Study on Inheritance and Its Type in Object Oriented Programming Using JAVA
- To understand object-oriented principles like abstraction, encapsulation, inheritance, polymorphism and apply them in solving problems
- Describe the benefits of the Object-Oriented programming approach

Research Methodology:

The study's findings are based on secondary data gathered from credible sources such as publications, books, magazines, and the internet. The research design for the study is primarily descriptive. Readings from journals These reputable articles were discovered using search engine platforms such as Google Scholar, global economics and business journals, open educational materials, and other popular websites.

Result and Discussion:

Types of Inheritance in Java

- Single Inheritance
- Multi-level Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

1. Single Inheritance

In Java, single inheritance refers to a subclass that extends only one superclass. Here's an example that demonstrates single inheritance.

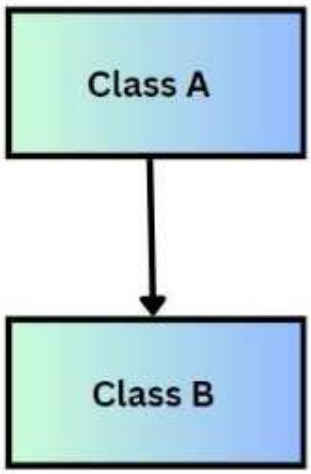

Figure 1: Single Inheritance

2. Multi-level Inheritance

In Java, multi-level inheritance refers to a scenario in which one class inherits properties and behaviors from another, which then inherits from another. This generates a hierarchical system of classes, with each class inheriting from the one above it. [9]

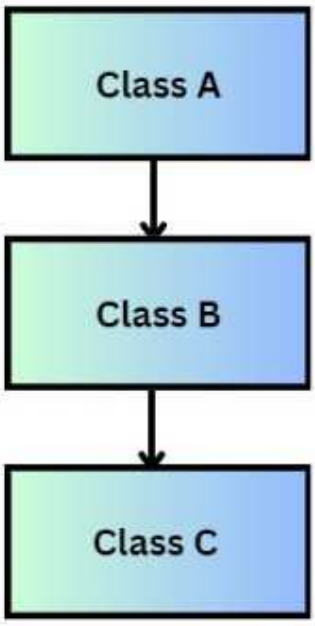

Figure 2: Multi-level Inheritance

3. Hierarchical Inheritance

In Java, hierarchical inheritance refers to how several classes inherit properties and behaviors from a single parent class. This inheritance structure consists of a single parent class and numerous child classes that inherit from it. [10]

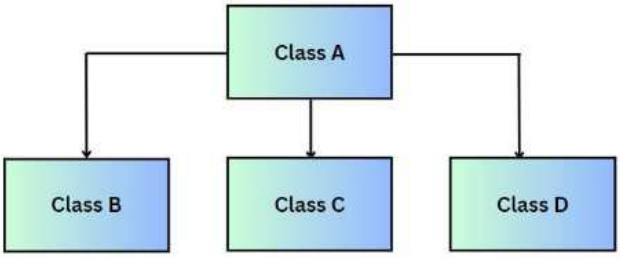

Figure 3: Hierarchical Inheritance

4. Hybrid Inheritance

Hybrid inheritance in Java is a blend of multiple inheritance and hierarchical inheritance. In hybrid inheritance, a class is derived from two or more classes, each of which can have its own subclasses. Due to the diamond problem, Java does not natively enable multiple inheritance. However, hybrid inheritance can be produced by combining hierarchical inheritance and interface implementation.

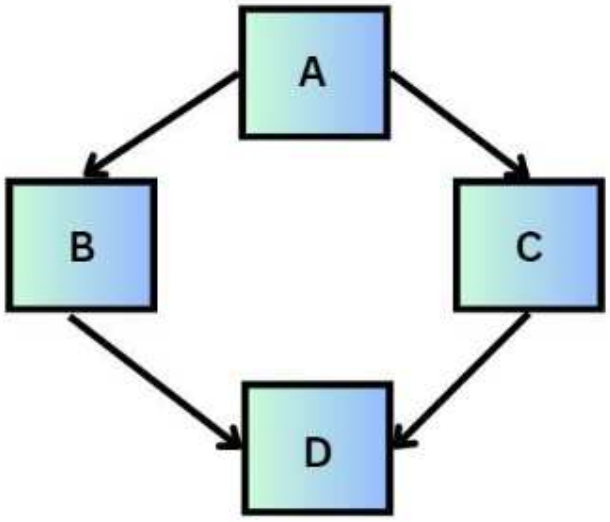

Figure 4: Hybrid Inheritance

Object-Oriented Programming in Java:

Object-Oriented Programming (OOP) is a programming paradigm that employs objects and classes to generate models of the real world. Java, being an object-oriented programming language, uses these notions to give a clear modular framework for applications. [11]

OOPs concepts in Java:

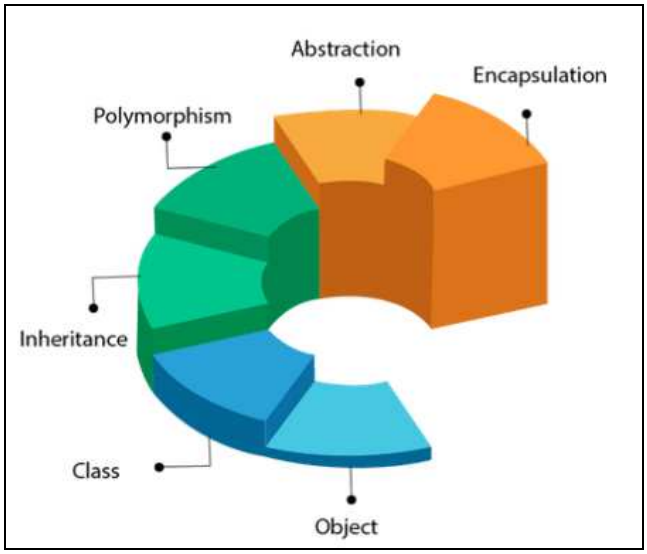

Figure 5: OOPs concepts in Java

- Class
- Object
- Encapsulation,
- Inheritance,
- Polymorphism,
- Abstraction,

Class: A class is a user-defined data type that includes data members and member functions for manipulating those data members. It is a group of related types of objects. A class is a general description of an object. It is the blueprint of an object. Class is an extension of the structure used in the C language. In the structure, we can merge multiple data elements into a single entity. In the class, we can combine various data elements and member functions. Class is a user-defined data type that allows us to specify both variables and functions. Class is the most important aspect of object-oriented programming. The class implements encapsulation, data abstraction, and data hiding.

Object: Objects are the fundamental runtime elements in object-oriented programming. Each object has data and code for manipulating that data. Objects can interact without needing to comprehend certain statistics or code. In structured programming, problems are tackled by separating them into functions. In contrast, object-oriented programming separates the problem into objects. Thinking in terms of objects rather than functions simplifies programming design.

Encapsulation

Encapsulation is the process of combining data (variables) and code (methods) that alter the data into

a single entity known as a class. It also limits direct access to parts of an object's components, which helps to prevent inadvertent interference and misuse of methods and data.

Java Example of Encapsulation:

```
public class Employee {
    private String name;
    private int age;

    public void setName(String newName) {
        name = newName;
    }

    public String getName() {
        return name;
    }

    public void setAge(int newAge) {
        age = newAge;
    }

    public int getAge() {
        return age;
    }
}
```

Inheritance in Java

In Java, inheritance is a process by which one class obtains another class's properties and behaviors (methods). This facilitates code reuse and method overriding. A subclass inherits from a superclass.

Java Example of Inheritance:

```
class Animal {
    void eat () {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}

public class TestInheritance {
    public static void main(String args[]) {
        Dog d = new Dog();
        d.bark();
        d.eat();
    }
}
```

Polymorphism in Java

Polymorphism in Java enables objects to be viewed as instances of their parent class rather than their own class. The most common application of polymorphism in OOP is when a parent class reference refers to a child class object.

Java Example of Polymorphism:

```
class Bird {
    void sing() {
        System.out.println("Bird is singing");
    }
}

class Sparrow extends Bird {
    void sing() {
        System.out.println("Sparrow is singing");
    }
}

public class TestPolymorphism {
    public static void main(String args[]) {
        Bird b;
        b = new Sparrow(); // Polymorphism
        b.sing();
    }
}
```

The Concept of Abstraction in Java

Abstraction is the process of masking implementation details and presenting simply functionality to the user. Abstraction in Java is performed using abstract classes and interfaces. [12]

Java Example of Abstraction;

```
abstract class Shape {
   abstract void draw();
}
class Rectangle extends Shape {
   void draw() {
      System.out.println("drawing rectangle");
   }
}
class Circle extends Shape {
   void draw() {
      System.out.println("drawing circle");
   }
}
public class TestAbstraction {
   public static void main(String args[]) {
      Shape s = new Circle(); // In real scenario, object is provided through method, e.g., getShape() method
      s.draw();
   }
}
```

Conclusion:

Java, like any other programming language design, is an experiment. Unlike most language designs, the extensive adoption of Java, and the resulting widespread availability of sizable "real-world" Java programs, allows us to ultimately evaluate that experiment in ways that most other languages simply cannot. Object-Oriented Programming (OOP) is critical in Java and other aspects of software development. Embracing OOP principles such as modularity, encapsulation, inheritance, polymorphism, and abstraction allow developers to design modular, maintainable, and scalable Java applications.

References

[1] Cook, William R.; Hill, Walter; Canning, Peter S. (1990). "Inheritance is not subtyping". POPL '90: Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 125–135.doi:10.1145/96709.96721. ISBN 0- 89791-343-4.

[2] Procedure Oriented Programming (POP) vs Object Oriented Programming (OOP). 2011.

[3] Bertrand Meyer (2009). Touch of Class: Learning to Program Well with Objects and Contracts. Springer Science & Business Media. p. 329.

[4] Kindler, E.; Krivy, I. (2011). "Object-Oriented Simulation of systems with sophisticated control". International Journal of General Systems. 40 (3): 313–343.

[5] Bertrand Meyer (2009). Touch of Class: Learning to Program Well with Objects and Contracts. Springer Science & Business Media. p. 329.

[6] Hachadi, Z. Java Web Development Springboot Security. (LinkedIn,2023), https://tn.linkedin.com/posts/zakaria-hachadi-176b871b0java – webdevelopment –springboot – activity – 7047332925712785408 – x4Y 0

[7] Chidamber, S., Darcy, D., Kemerer, C.: Managerial use of metrics for object-oriented software: an exploratory analysis. IEEE Trans. Software Engineering 24(8), 629–639 (1998)

[8] Daly, J., Brooks, A., Miller, J., Roper, M., Wood, M.: Evaluating inheritance depth on the maintainability of object-oriented software. Empirical Software Engineering 1(2), 109–132 (1996)

[9] V. T. Lokare, P. M. Jadhav, and S. S. Patil, ―An integrated approach for teaching object-oriented programming (C++) course, ‖ Journal of Engineering Education Transformations, vol. 31, no. 3, pp. 17–23, 2018.

[10] R. Fojtík, ―Teaching of object-oriented programming, ‖ in Proc. the 12th International Scientific Conference on Distance Learning in Applied Informatics., 2018, pp. 273–282.

[11] kaur, l., kaur, n., ummat, a., kaur, j., & kaur, n. (2016). research paper on object-oriented software engineering. international journal of computer science and technology, 36-38.

[12] Lafore, Robert, Object-Oriented Programming in C++, Fourth Edition, Sams Publishing, 2002. ISBN 0-672- 32308-7.