International Journal on Science and Technology (IJSAT)

E-ISSN: 2229-7677 ● Website: www.ijsat.org ● Email: editor@ijsat.org

A Study On Abstraction in Object Oriented Programming with Java and Its Implementation in Developing Programs

Mahesh Lavand[1], Dr. Rupali Pawar[2]

[1]Student MCA, [2]Asst. Professor, MCA Dept.
[1,2]Zeal Institute of Business Administration, Computer Application & Research, Pune-41

Abstract

Object-Oriented Programming (OOP) has revolutionized the way software development is approached by promoting modularity, reusability, and abstraction. Java, one of the most widely used programming languages in the world, is inherently object-oriented, providing a robust platform for the development of scalable and maintainable applications. This paper explores the key principles of OOP, highlights Java's support for these principles, and examines the strengths and challenges of using Java for object-oriented development. Additionally, the paper discusses the evolution of Java in terms of OOP features and how modern Java applications leverage these principles for efficient, modular, and reusable code.

This paper focuses on the abstraction concept in Object Oriented Programming implementation in Java programming language.

Keywords: Class, Object, Abstraction, OOP, Java

1. Introduction

Abstraction is one of the core concepts of Object-Oriented Programming (OOP) in Java. Abstraction is the concept of defining a method in one class and implementing it in a subclass. It hides the implementation details and only shows the essential features. Abstraction is used to design and scalable structure of code and clear separation between functionality and implementation.

Abstraction is the mechanism of hiding the implementation details and showing only functionality to the user. It allows the focus to be on what an object does rather than how it does it. This is achieved by two ways using abstract classes and interfaces.

A class which is declared by using abstract keywords is known as abstract class that cannot be instantiated. It can have abstract methods (method without a body) and non-abstract methods. It needs to be extended in another class and must implement abstract methods.

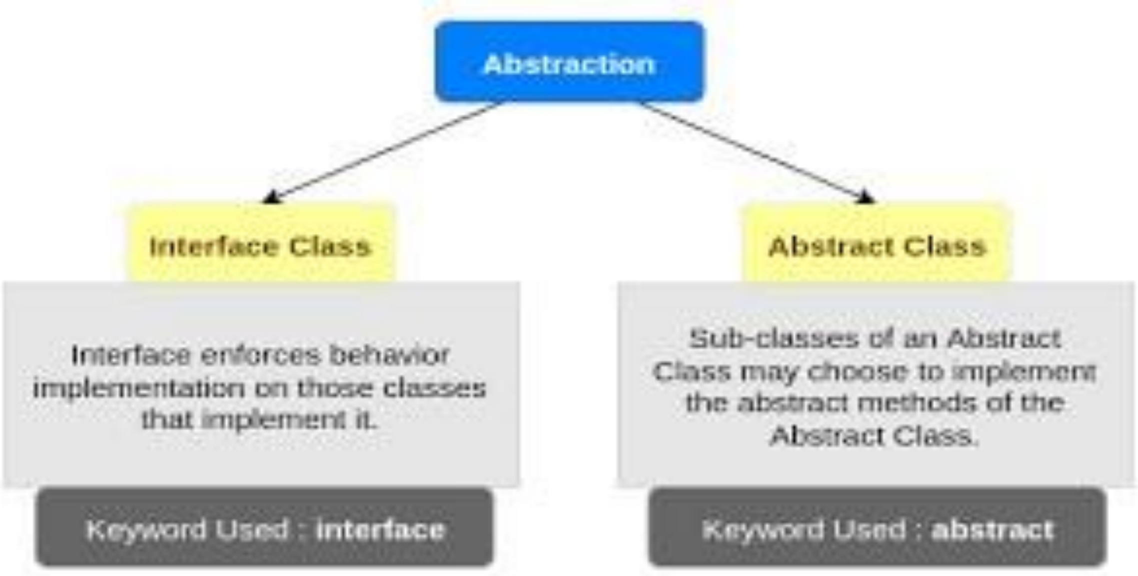
An interface in java is a blueprint of a class; it has static, constant and abstract methods only. It is used to achieve full abstraction and multiple inheritance in java.

For example, when you drive a car, you only use the steering wheel, brakes, and accelerator. You don't need to understand how the engine or other parts work. Abstraction works the same way it hides the complicated details and shows you only the important things you need to use.

If applications are developed using the concept of Abstraction, they have the following advantage

- **Simplifies Complexity** – Hides implementation details and exposes only essential functionalities, making the system easier to understand.
- **Enhances Maintainability** – Changes in the internal implementation do not affect other parts of the application, reducing maintenance efforts.
- **Improves Code Reusability** – Abstract classes and interfaces allow developers to reuse code across different modules or applications.
- **Supports Scalability** – Developers can extend functionality without modifying existing code, making the application more scalable.
- **Increases Security** – Restricts direct access to implementation details, reducing the risk of accidental modification or misuse.
- **Encourages Modular Design** – Breaks the application into smaller, manageable components, improving development and testing.

Abstraction:



Abstract class:

An abstract class is a class that is declared using the abstract keyword. It can have both abstract and non-abstract methods. Abstract methods do not have a body and must be implemented by subclasses. An abstract class cannot be instantiated directly; instead, it needs to be extended by another class that provides implementations for its abstract methods. We can use an abstract class, but there is no guarantee of achieving 100% abstraction.

Key points about abstract class:

- An abstract class must be declared using the abstract keyword.
- It can contain both abstract and non-abstract methods.
- It cannot be instantiated directly.
- You can create a reference variable of an abstract class

- It can have constructors and static methods.
- It can have final methods, which prevent subclasses from changing the method's implementation.

Syntax:

```
access_specifier abstract class class_name {
}
```

Example:

```
package demoPrograms;
abstract class Car {     abstract void
startEngine();     void applyBrakes()
{
        System.out.println("Brakes applied. Car is slowing down...");
    }
} class Tesla extends Car {
    @Override                 void
startEngine() {
        System.out.println("Tesla engine started silently. Ready to drive!");
    }
} class Ferrari extends Car {
    @Override                 void
startEngine() {
        System.out.println("Ferrari engine roars to life! Vroom Vroom!");
    }
} public class Main {     public static void
main(String[] args) {        Car t = new Tesla();
        t.startEngine();
        t.applyBrakes();        Car f =
new Ferrari();
        f.startEngine();
        f.applyBrakes();
```

```
    }
}
```

Abstract Method:

An abstract method is a method declared without any implementation or method declared but not defined. That means the body of these methods are omitted. It is called an abstract method. The method's body (implementation) is provided by the subclass. Abstract methods can never be final or static.

If a class extends an abstract class, it must provide code for all the abstract methods of the parent class. But if the subclass is also abstract, it doesn’t have to implement those methods right away. The next subclass will handle it. An abstract method is just a method with a name and parameters but no code inside (no body).

The abstract methods are declared using the keyword abstract. The abstract methods should be defined in the subclass. An abstract method does not have any body. It always ended with a semicolon (;).

Syntax: accessSpecifier abstract returnType methodName(arguments);

Interface:

An interface in Java is a blueprint of a class. It has static, constant, and abstract methods only. It is used to achieve full (100%) abstraction and multiple inheritance.

In other words, interfaces primarily define methods that other classes must implement. The Java interface also represents the Is-A relationship.

A class can implement more than one interface. In Java, an interface is not a class but a set of requirements for the class that we want to conform to the interface.

All the methods of an interface are by default public, so it is not required to use the public keyword when declaring a method in an interface.

Interfaces can also have more than one method. Interfaces can also define constants but do not implement methods.

There are mainly three reasons to use an interface:

1. It is used to achieve full abstraction.
2. By using an interface, we can support multiple inheritance.
3. It can be used to achieve loose coupling.

An interface is similar to a class, but it is a collection of public static (final) variables (constants) and abstract methods.

Syntax:

```
access_specifier interface Interface_Name {
    Return_Type method1(parameters);
```

```
    Type variable_name = value;
}
```

Static method in interface:

Java 8 provides the feature to define static methods in an interface. It allows defining static methods with a body (implementation) inside the interface. These methods are defined like normal methods but use the static keyword in the method signature. They do not need to be extended or implemented in a class. Static methods cannot be overridden in any class because they belong to the interface itself.

Advantages of Static Methods in an Interface

- Static methods can be written inside an interface, keeping related functions together.
- No need to create extra helper or utility classes.
- They are called using the interface name, so there’s no confusion with other methods.
- Saves time by reusing the same method instead of writing it again in different classes.
- Makes the code neat and well-organized by keeping useful functions in one place.

- **Example static method in an interface in Java**

```
 interface CurrencyConverter
{
public static double convertUsdToInr(double usd)
{
            double exchangeRate = 83.0;
double inr = usd * exchangeRate;
            return inr;
    }
}
public class StaticMethodTest
{
public static void main(String args[])
{
```

```
        double usdAmt = 30;

        double inrAmt = CurrencyConverter.convertUsdToInr(usdAmt);

        System.out.println(usdAmt + " USD = " + inrAmt + " INR");

    }

}
```

Access Modifiers in Java

There are four types of access modifiers:

1. **Private:**
   The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:**
   The access level of the default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be default.
3. **Protected:**
   The access level of a protected modifier is within the package and outside the package through child class (inheritance). If you do not make the child class, it cannot be accessed from outside the package.
4. **4) Public:**
   The access level of the public modifier is everywhere. It can be accessed from within the class, outside the class within the package, and outside the package.

Access Modifier	Within Class	Within Package	Outside Package by Subclass Only	Outside Package
Private	Yes	No	No	No
Default	Yes	Yes	No	No
Protected	Yes	Yes	Yes	No
Public	Yes	Yes	Yes	Yes

2. Conclusion

Abstraction is a fundamental concept in Object-Oriented Programming (OOP) that enhances software design by simplifying complexity, improving maintainability, and increasing code reusability. By using abstract classes and interfaces, Java allows developers to define essential functionalities while hiding implementation details. Abstract classes enable partial abstraction by allowing both abstract and concrete methods, whereas interfaces provide full abstraction and support multiple inheritance.

Through abstraction, Java promotes a scalable, modular, and secure coding structure, ensuring that programs are easier to understand, extend, and maintain. Additionally, Java 8 introduced static methods in interfaces, further enhancing code organization and reusability.

By implementing abstraction effectively, developers can build efficient, flexible, and maintainable applications, ensuring clear separation between interface and implementation. This study highlights the significance of abstraction in Java and its role in developing robust and scalable software systems.

References

1. 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.
2. Kak, Avinash C. Programming with Objects, A Comparative Presentation of Object-Oriented Programming with C++ and Java, John Wiley, 2003. ISBN 0-471-26852-6.
3. Lafore, Robert, Object-Oriented Programming in C++, Fourth Edition, Sams Publishing, 2002. ISBN 0-672 32308-7. [4] Seed, Graham M., An Introduction to Object-Oriented Programming in C++ with Applications in Computer Graphics, Second Ed., Springer-Verlag, 2001.ISBN 1 85233-450-9.
4. Svenk, Goran, Object-Oriented Programming: Using C++ for Engineering and Technology Delmar, 2003. ISBN 0-7668-3894-3. [
5. Yevick, David, A First Course in Computational Physics and Object-Oriented Programming, Cambridge University Press, 2005. ISBN 0-521-82778-7.
6. Amit Verma, Navdeep Kaur Gill,"Image Processing and Watermark", International Journal of Computer
7. NAYER A ,F F WU and K IMHOT Object Oriented Programming for Flexible software Example of a Load flow Power Systems IEEE Transactions on 1990 5(3) :Page 689-696
8. Katura : Object Oriented Programming and Software Engineering 1996
9. Procedure Oriented Programming(POP) vs Object Oriented Programming(OOP). 2011.
10. Zimmerman, M.L. and P. Mallasch. Using object-oriented Programming in computational electromagnetic codes. in Antennas and Propagation Society International Symposium, 1995. AP-S. Digest.1995.
11. Omar N,N A Razik Determining the basic elements of Object Oriented Programming using Natural Language Processing in International Technology 2008
12. McIntyre, S.C. and L.F. Higgins, Object-oriented Systems Analysis and Design: Methodology and Application. Journal of Management Information Systems, 1988. 5(1): p. 25-35.
13. Horstmann, C. S. (2014). Core Java Volume I: Fundamentals. Prentice Hall.
14. Bloch, J. (2008). Effective Java (2nd Edition). Addison-Wesley.
15. Koller, D. (2000). Java 2: The Complete Reference. McGraw-Hill.
16. Gosling, J., Joy, B., & Steele, G. (2005). The Java Language Specification (3rd Edition). Addison-Wesley.
17. Schildt, H. (2018). Java: The Complete Reference (11th Edition). McGraw-Hill.