COMPARATIVE ANALYSIS OF OBJECT-ORIENTED PROGRAMMING IN PYTHON AND JAVA

Author: **Nomozova Malika**
Course: Object Oriented Programming
Instructor: **Muslimbek Pirnazarov**
University: Millat Umidi University
Date: 25/04/2026

Abstract

Python and Java are two of the most popular programming languages used today, and both support Object-Oriented Programming. But anyone who has worked with both knows they feel completely different. This paper looks at how each language implements the four core OOP concepts — encapsulation, inheritance, polymorphism, and abstraction — and tries to understand why those differences exist and what they mean in practice. The comparison is based on code examples and existing literature rather than performance benchmarks. The conclusion is straightforward: neither language is better than the other. They just make different trade-offs, and knowing those trade-offs helps you pick the right tool for the right job.

1. Introduction

If you have spent any time learning programming, you have probably used Python or Java — or both. They show up everywhere: university courses, job listings, open-source projects, big companies. What is interesting is that both languages support Object-Oriented Programming, yet writing OOP code in Python feels nothing like writing it in Java.

OOP changed how we build software. Instead of writing a long list of instructions, you think in objects — things that have their own data and their own behavior. A user account, a bank transaction, a product in a store — all of these can be modeled as objects. It makes code easier to organize, easier to reuse, and easier to maintain over time.

So why compare Python and Java? Because they represent two very different answers to the same question: how much should a language control the way you write code? Java is strict. It wants everything defined upfront and enforces its rules at compile time. Python is relaxed. It gives you the tools and trusts you to use them well. Both approaches work — but they work differently depending on what you are building.

This paper compares how Python and Java handle the four core OOP principles: encapsulation, inheritance, polymorphism, and abstraction. The goal is not to pick a

winner, but to understand what each language does well and where each one makes things harder than they need to be.

2. Literature Review

Developers have been comparing Python and Java for years. The debate does not get old because both languages keep evolving and the way people use them keeps changing.

One of the first things that comes up in the literature is the difference in design philosophy. Java was built from the start as a pure object-oriented language — everything must be inside a class, no exceptions. Python is different. You can write a complete Python program without ever defining a single class, and that is perfectly acceptable. Several sources point to this as the root of most other differences between the two languages.

Syntax is another common topic. Studies consistently show that Python requires far fewer lines of code to accomplish the same task. One well-known example compares reading a text file: Python does it in about four lines, Java needs closer to eighteen. For beginners, this gap makes Python feel natural and Java feel unnecessarily complicated. But some argue that Java's verbosity forces developers to think more carefully about what they are writing, which builds better habits over time.

When it comes to specific OOP features, the differences become more technical. On encapsulation, Java provides strict access modifiers that are enforced by the compiler. Python uses naming conventions that signal intent but do not actually prevent access. On inheritance, Python supports multiple inheritance natively while Java limits classes to a single parent and uses interfaces for the rest. On abstraction, Java has built-in support through abstract classes and interfaces, while Python offers the same through its abc module — but only if you choose to use it.

Performance comparisons have produced some surprising results. In certain tests involving large numbers of objects, Python has matched or slightly outperformed Java. Most experts still consider Java faster for large-scale systems, but the gap is smaller than many people assume.

The overall picture from the literature is clear: Python is the preferred choice for data science, machine learning, rapid development, and smaller projects. Java dominates in enterprise software, Android development, and systems where long-term reliability matters most.

3. Methodology

This paper uses a qualitative comparative approach. Rather than running benchmarks or collecting survey data, the analysis focuses on how each language implements OOP concepts through code examples and documentation.

The four concepts chosen — encapsulation, inheritance, polymorphism, and abstraction — were selected because they are the foundation of OOP in any language.

Understanding how a language handles these four ideas tells you most of what you need to know about its OOP design.

For each concept, the structure is the same: a short explanation of what the concept means, a code example in both Python and Java, and a discussion of what the differences mean for developers. Sources include the official documentation for both languages, as well as technical articles from Real Python, DataCamp, and InformIT. Where opinions appear, they are presented as such, not as facts.

4. Comparison

4.1 Encapsulation

Encapsulation means keeping your data protected and only exposing what needs to be exposed. Both languages support this idea, but they enforce it very differently.

Java uses access modifiers — private, protected, and public. If a variable is marked private, nothing outside the class can access it directly. The compiler makes sure of that. You write getters and setters to control access:

java

```
public class BankAccount {
    private double balance;

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
}
```

Python does not have real access modifiers. A single underscore before a variable name means "please don't touch this," and a double underscore makes it harder to access through name mangling — but not impossible. Python relies on developers following the convention rather than enforcing it:

python

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

    def get_balance(self):
```

```
        return self.__balance

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
```

The code looks similar, but the underlying attitude is different. Java *enforces* encapsulation. Python *suggests* it. For large teams, Java's approach reduces accidents. For smaller projects, Python's approach is fast and practical.

4.2 Inheritance

Inheritance lets one class build on top of another, reusing and extending its behavior. Both languages support it, but Java only allows a class to inherit from one parent. Python allows multiple.

Java uses the extends keyword and keeps things simple and predictable:

```
java
public class Animal {
    public void speak() {
        System.out.println("Some sound...");
    }
}

public class Dog extends Animal {
    @Override
    public void speak() {
        System.out.println("Woof!");
    }
}
```

Python allows a class to inherit from more than one parent at the same time. When two parents share a method name, Python resolves the conflict using the Method Resolution Order:

```
python
class Animal:
    def speak(self):
        print("Some sound...")

class Pet:
    def is_domestic(self):
        return True

class Dog(Animal, Pet):
```

```
    def speak(self):
        print("Woof!")
```

Python's multiple inheritance is more powerful, but it requires the developer to understand how conflicts get resolved. Java's single inheritance is more limiting, but the code is easier to follow. It comes down to whether you prefer flexibility or clarity.

4.3 Polymorphism

Polymorphism means different objects can respond to the same method call in their own way. It is one of the most useful features of OOP, and it is also one of the clearest examples of how different Python and Java really are.

Java achieves polymorphism through inheritance and method overriding. The compiler checks types and relationships before anything runs:

```
java
public class Shape {
    public double area() {
        return 0;
    }
}

public class Circle extends Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    @Override
    public double area() {
        return Math.PI * radius * radius;
    }
}

public class Rectangle extends Shape {
    private double width, height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
```

```
    @Override
    public double area() {
        return width * height;
    }
}
```

Python uses duck typing. If an object has the method you are calling, Python runs it — no formal relationship required:

```
python
class Circle:
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return 3.14159 * self.radius ** 2

class Rectangle:
    def __init__(self, width, height):
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

shapes = [Circle(5), Rectangle(4, 6)]
for shape in shapes:
    print(shape.area())
```

Circle and Rectangle share no parent class here, yet the loop works perfectly. Python does not care about formal relationships — it just checks whether the method exists when it needs to call it. Java catches problems earlier. Python catches them later but writes less code to get there.

4.4 Abstraction

Abstraction means hiding complex implementation details and only showing what is necessary. It is the most conceptual of the four principles, and also the one where Python and Java feel most different.

Java has abstraction built directly into the language. Abstract classes cannot be instantiated on their own, and any child class must implement all abstract methods — the compiler enforces this:

```
java
public abstract class Animal {
```

```
    public abstract void speak();

    public void breathe() {
        System.out.println("Breathing...");
    }
}

public class Dog extends Animal {
    @Override
    public void speak() {
        System.out.println("Woof!");
    }
}
```

Python supports the same concept through the abc module, but you have to opt in:

```
python
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

    def breathe(self):
        print("Breathing...")

class Dog(Animal):
    def speak(self):
        print("Woof!")
```

The result is the same, but the path is different. Java puts abstraction in your face from the beginning. Python makes it available but does not push you toward it. You can write entire Python applications without ever using the abc module. In Java, you will encounter abstract classes whether you want to or not.

5. Results and Discussion

After going through all four OOP concepts, a clear pattern shows up: Java enforces good structure, Python encourages it. This is not a coincidence — it reflects a deliberate design choice made by the people who built each language.

Java catches more mistakes before the program runs. Its compiler checks types, access levels, and class relationships upfront. This is genuinely useful when you are working on a large system with many developers who did not all write the same code.

Python catches mistakes at runtime instead. This makes development faster in the early stages but can lead to bugs that only show up when a specific part of the code gets executed.

Python's polymorphism through duck typing is arguably its strongest OOP feature. The ability to use objects interchangeably without formal class hierarchies leads to clean, readable code that does not feel bloated. Java's approach produces more code, but every part of that code is explicit and verifiable.

On abstraction, Java's enforced contracts between classes are valuable in long-lived codebases. When a new developer joins a project and sees an abstract class, they immediately know what any subclass is required to implement. Python's optional approach works well when the team is small and everyone knows the codebase, but it requires more discipline to maintain as the project grows.

The choice between these two languages ultimately comes down to context. Java makes more sense for enterprise applications, large teams, and systems that need to run reliably for years. Python makes more sense for data science, machine learning, rapid prototyping, and projects where getting something working quickly matters more than enforcing strict structure. Neither is the wrong choice — they are just different tools for different situations.

6. Conclusion

This paper compared how Python and Java implement encapsulation, inheritance, polymorphism, and abstraction. The technical differences are real and significant, but the more interesting finding is what those differences reveal about each language's underlying philosophy.

Java trusts structure over people. It enforces rules at every level because it assumes that, in a large enough system, someone will eventually make a mistake if the language lets them. Python trusts people over structure. It assumes that developers are capable of making good decisions and gives them the freedom to do so.

Both assumptions are reasonable. Both lead to good software when applied in the right context. The mistake is treating one language as universally better than the other.

For anyone learning programming, studying both Python and Java is genuinely worthwhile. Python teaches you to write code that is clean and expressive. Java teaches you to write code that is structured and deliberate. The skills you build from each language complement each other in ways that make you a better developer overall.

References:

1. Python Software Foundation. (2024). *Python Documentation — Classes*. https://docs.python.org/3/tutorial/classes.html

2. Oracle Corporation. (2024). *Java Documentation — Object-Oriented Programming Concepts*. https://docs.oracle.com/javase/tutorial/java/concepts/

3. Real Python. (2024). *Object-Oriented Programming in Python vs Java*. https://realpython.com/oop-in-python-vs-java/

4. DataCamp. (2024). *Python vs Java: Differences and Similarities in 9 Key Areas*. https://www.datacamp.com/blog/python-vs-java

5. Morris, S. (2021). *Comparing Python Object-Oriented Code with Java*. InformIT. https://www.informit.com/articles/article.aspx?p=2436668

6. Chalasani, V. (2025). *Why Python is Better than Java for OOP*. Medium. https://medium.com/@vedasree812/why-python-is-better-than-java-for-object-oriented-programming

7. ActiveState. (2021). *Java versus Python: Key Programming Differences*. https://www.activestate.com/blog/java-versus-python-key-programming-differences-in-2021/

8. Prasanthi, K. (2025). *Python vs Java — Part 2: OOP Differences*. Medium. https://medium.com/@kslprasanthi/python-vs-java-part-2-oop-differences-beginner-friendly-guide