Here's a detailed tabular comparison of **C++**, **Python**, and **Java** based on key aspects of their object-oriented nature and features:

Feature	C++	Python	Java
OOP Support	Strong support, but multi-paradigm (supports OOP, procedural, and generic programming)	Purely OOP (supports multiple paradigms, but OOP is the default)	Strictly OOP (except for primitives)
Everything is an Object?	No (Primitives are not objects)	Yes (Everything, including primitive types, is an object)	No (Primitives are not objects, but objects are central)
Procedural Programming	Allowed and commonly used (no OOP required)	Allowed but less common in practice	Not supported (everything must be in a class)
Inheritance	Supports single and multiple inheritance	Supports single inheritance, mixins for multiple inheritance	Supports single inheritance only (through classes)
Memory Management	Manual memory management (using `new`/`delete`)	Automatic garbage collection (GC)	Automatic garbage collection (GC)
Typing	Static typing (explicit variable types required)	Dynamic typing (no need for type declarations)	Static typing (explicit variable types required)
Syntax Complexity	Complex, with more detailed syntax and requirements	Simple and readable syntax, less verbose	Simple and structured, but more verbose than Python
Class and Object Creation	Explicit `class` declaration, `new` operator for objects	Classes are created and used in the same way, but no need for `new`	Classes must be declared and objects created with `new`
Access Modifiers	Supports `public`, `private`, `protected` access modifiers	Supports `public` and `private` (by convention, but can be accessed due to Python's dynamic nature)	Supports `public`, `private`, `protected`, and `default` (package-private)
Exception Handling	Supports exception handling (`try-catch`)	Supports exception handling (`try-except`)	Supports exception handling (`try-catch`)
Method Overloading	Supported (by function signature)	Not supported (can use default arguments for similar behavior)	Supported (by method signature)
Operator Overloading	Supported	Not supported	Not supported

Garbage Collection	No built-in garbage collection (manual memory management)	Automatic garbage collection (via `gc` module)	Automatic garbage collection (via JVM)
Concurrency	Threads are managed manually with libraries like `thread` or `pthread`	Threading supported via `threading` module, but simpler than C++	Concurrency managed via `Thread` class, `ExecutorService`, etc.
Compilation	Compiled language (with the need for linking)	Interpreted, can be compiled using tools like `PyInstaller`	Compiled to bytecode, runs on the JVM
Interpreted vs Compiled	Compiled language	Interpreted language	Compiled language (to bytecode for JVM)
Functional Programming	Limited (via lambda expressions, `std::function`, etc.)	Strong support (first-class functions, lambdas)	Limited (via lambdas introduced in Java 8)
Runtime Environment	Native execution (directly on the machine)	Interpreted or compiled to binary (via tools)	Runs on JVM (platform-independent)
Libraries/Frameworks	Rich ecosystem of libraries and frameworks	Vast number of libraries, especially for data science and scripting	Extensive libraries and frameworks for enterprise applications (e.g., Spring)
Performance	Very high (due to low-level memory control)	Slower than C++ due to dynamic nature	Slower than C++ but faster than Python due to JVM optimizations
Portability	Platform dependent (native compilation for each platform)	Platform-independent (Python interpreter)	Highly portable (JVM runs on any platform)
Compilation Speed	Slower compilation time (due to manual optimization and linking)	Fast (since it's interpreted)	Moderate (compiles to bytecode)

Summary:

1. **C++** is a **multi-paradigm** language, highly flexible but **not purely OOP**. It allows procedural, object-oriented, and generic programming. It has manual memory

management and complex syntax but offers high performance and fine control over system resources.

2. **Python** is **purely OOP** in its design, with everything being an object (including primitive types). It offers simple and readable syntax with dynamic typing, automatic garbage collection, and strong support for both OOP and functional programming. However, it is slower than C++ and Java due to its interpreted nature.
3. **Java** is a **strictly object-oriented language** (except for primitive types) and runs on the JVM. It has a highly structured approach to OOP with enforced class-based design. It supports automatic garbage collection and provides a strong, statically-typed system. Its performance is generally better than Python but slower than C++.

Each of these languages has its own strengths, and the best choice depends on the application and the specific requirements you have, such as performance, ease of use, or ecosystem.