# Object-Oriented Languages: A Comparison

The following tables compare four major O-O languages: Eiffel, C++, C#, Java and Smalltalk.

Although this comparison is part of the Eiffel pages, its intent is to provide a balanced coverage and to generate light, not heat. The two components are clearly separated:

- News, in the form of factual comparisons for each property. (If you find any error, please report it so that we can correct it.)
- Our comments, marked by a green bullet.

If you find the table useful, please put a link to it in your own pages.

The references to Eiffel without further qualification apply to the language. Comments about the environment apply to ISE Eiffel, the leading implementation.

We hope that you find this comparison useful and that it helps you make your choice based on facts, not passion.

## Design by Contract™, assertions

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Design by Contract: language and environment support assertions | Nothing comparable (only an "assert" instruction) | Nothing comparable (only an "assert" instruction). Various non-standard assertion proposals. | Nothing comparable |
| **Our comment:** The idea of Design by Contract is a key tool for producing reliable software, for documentation, and for debugging. Only Eiffel implements it through assertions (preconditions, postconditions, invariants) in the language itself, associated documentation tools, and automatic mechanisms for testing the assertions. | | | |

## Static typing

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Statically typed | Statically typed; however the language supports C-style "casts" which really amount to invitations to violate type rules | Typed, mostly statically but dynamic typing is required for generic container structures. | Dynamically typed |
| **Our comment:** Static typing enhance readability and enables compilers to catch errors long before they can become a run-time problem; this saves development time, saves money, and yields producing more reliable code. | | | |

## Proprietary status, vendor interoperability

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Open, multi-vendor, standardized | Open, multi-vendor (ANSI standard) | Multi-vendor, licensed from Sun | Multi-vendor, fairly wide variation between vendors |
| **Our comment:** Although C++ has the prestige of an ANSI standard, practical interoperability is far from perfect, especially since the standard has been evolving so quickly, and implementations such as Visual C++ depart from the standard in a number of areas. Smalltalk experts also report that it is hard to move from an implementation to another. For Java, Sun has refused to relinquish the trademark and has insisted on remaining in control of the language, infuriating many other companies.<br><br>Eiffel has been stable for several years and there is a Kernel Library standard (ELKS 95), controlled, as the language, by the NICE consortium, the standards body for Eiffel. | | | |

## Compilation technology

| ISE Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Combination of interpretation and compilation in same environment | Usually compiled | Usually mix of interpretation and "on-the-fly" compilation | Historically interpreter-based, currently mix of interpretation and compilation |

**Our comment:** Compilation is needed for run-time efficiency; interpretation yields compilation efficiency. We think it is hard to match ISE Eiffel's combination of a quick compilation mechanism -- the Melting Ice Technology (TM) -- for development and debugging, and extensively optimized "final compilation" for generation of efficient production code.

## Efficiency of generated code

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Generates fast executables. | Generates fast executables. | Widely reported performance problems. | Executables require a "Smalltalk image". |

**Our comment:** Eiffel, and ISE Eiffel in particular, are focused on performance. Benchmarks (particularly in array optimization) show them to be comparable in speed to hand-coded/optimized C code, with far less development effort and improved maintenance. C++ run-time performance is typically quite good. Performance, to put it diplomatically, is not Smalltalk's main claim to fame. Java users report performance problems similar to those of Smalltalk. ISE Eiffel has it both ways: bytecode generation for fast development; optimized compilation through C for fast execution.

## Automatic documentation

| ISE Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Documentation extracted automatically ("short", "flat-short" and other formats) without extra programmer effort. | No standard mechanism. | JavaDoc: add special comments. | No standard mechanism. |

**Our comment:** Eiffel introduced the idea that documentation should not be a separate product but a set of *views* extracted from the software itself by automatic tools. ISE Eiffel has a powerful set of mechanisms to generate many views, textual or graphical, from Eiffel source code: class interfaces, inheritance hierarchies, high-level "bubbles and arrows" diagrams etc. No extra work is required of the programer; Eiffel's syntactic clarity and built-in Design by Contract mechanisms play a key role here.

You can generate the result in many formats -- in fact, any format that you like, by defining a "filter" for that format using EFF, ISE's Eiffel Filter Format. Existing filters include HTML, Postscript, Microsoft Rich Text Format (Help files format), TeX, Troff, MML (for FrameMaker), ASCII. They can serve as models for in-house style rules or new formats.

The HTML format is particularly important as a way to publish a system architecture on an Intranet or Internet page and allow team collaboration on an ongoing design.

The risk is just too big, if you treat documentation as a product developed and maintained separately, that it will be incomplete or incorrect, especially as the software evolves (how do you guarantee that the documentation gets updated accordingly?). The designers of Java have acknowledged that the JavaDoc idea was imitated from Eiffel. JavaDoc goes in the right direction but requires you to add special comments, and does not have the benefit of the Design by Contract mechanisms. Other languages do not give you any standard facilities to produce documentation automatically as a derivative from the code. This is a major obstacle to obtaining some of the benefits of O-O development.

## Run-time fees

| ISE Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| No run-time fees. | No run-time fees. | No run-time fees for introductory versions, but 6-figure fees for embedded use. | No run-time fees (reversing previous policies of major vendor). |

**Our comment:** We think that run-time fees are harmful to the development of the software industry. There have been a number of press reports about the huge fees that Sun is exacting for the use of Java in embedded environments, and the industry's strong negative reactions. ISE Eiffel, in contrast, can be used without run-time fees; this is in the customers' best interest.

## Exception handling

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Exception handling | Exception handling | Exception handling | Exception handling |

**Our comment:** It seems everyone agrees here! Note that the Eiffel model, based on Design by Contract principles, goes further than the others by permitting a "rescue/retry" stragegy for error recovery.

## Multiple inheritance

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Multiple inheritance, widely used | Multiple inheritance (but use discouraged by many books and compiler vendors because of various problems). | Single inheritance (but multiple interface facility) | Single inheritance |

**Our comment:** Robust multiple inheritance facilities are essential to support combining various abstractions and reusable components, both in modeling (*CALCULATOR_WATCH* inherits from *CALCULATOR* and *WATCH*) and design/implementation (*INTEGER* inherits from *NUMERIC* and*COMPARABLE*). It seems just about everyone recognizes Eiffel's leadership here, with a careful design supporting renaming, "join", repeated inheritance etc.

Don't believe those who tell you multiple inheritance is tricky; that's true only in languages that don't support it, or don't support it well.

## Readability, clarity, ease of use

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Clear, simple, readable syntax. | Complex syntax | C++-like syntax. | Opinions differ. We feel it's pretty bizarre (right-association for operators, etc.) |

**Our comment:** Readability is essential to the development and maintenance of large systems with a long lifetime and many participants. Eiffel has been widely praised for the simplicity of the language and the clarity of the syntax, making it possible to teach the language quickly including to non-software-professionals (financial analysts, bankers, engineers...)

## Standardization of library style

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Libraries use standardized vocabulary | Wide variation - Libraries evolved before vocabulary was a recognized issue. | ? | Emphasis on consistency |

**Our comment:** The Eiffel emphasis on a strict set of standards for naming library components has proved a major boost to productivity and ease of learning. Even in Smalltalk a learning curve of six months is widely reported before people truly master the libraries.

## Automatic memory management

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Garbage Collection/ Automatic memory management | No garbage collection in common commercial implementations. | Garbage collection | Garbage collection |

**Our comment:** C++ is pretty much alone here, and the idea that programmers should be responsible for the tedious and error-prone task of reclaiming unused memory has fewer and fewer supporters.

ISE Eiffel appears to be unique in its combination of garbage collection and **multithreading**: in a multi-threaded context, each thread has its own garbage collector. This means that when a thread needs memory the others can proceed without disturbance. In other approaches that we have seen, when one thread needs garbage collection, all threads stop. This is unacceptable for embedded and all time-sensitive applications.

## Scope in the software lifecycle

| ISE Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Seamless Visual Development Environment | Addresses implementation only | Focus on implementation | Mostly focused on implementation |

**Our comment:** Among existing approaches, Eiffel is the only one that is not just an implementation language but covers the whole spectrum, from analysis and design to implementation and maintenance. Seamlessness and reversibility are the key terms. The benefits, technical and economic, are invaluable. "Design by Contract", used throughout the process, provides the unifying thread. You stay from beginning to end within the same concepts and tools, going as far forward or backward as you need without encountering the kind of "impedance mismatches" of people faced with, say, UML at one end and Java or Smalltalk at the other.

## Mathematical software

| ISE Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Libraries available | Libraries available | Libraries available or under development | Libraries available |

**Our comment:** Developers of high-end financial and scientific applications need to have comprehensive and high-performance mathematical/numerical libraries at their disposal. The EiffelMath libraries (based on the prestigious NAG libraries) provide high performance and well-abstracted concepts, making it easy for developers and non-technical users such as financial analysts to design and build complex applications easily.

## Openness and interoperability with legacy software

| Eiffel | C++ | Java | Smalltalk |
|---|---|---|---|
| Standard language and environment support for Integration with C and C++ | Good interoperability with C. | Native methods | No standardized C-language interface. |

**Our comment:** Clearly it is essential to interoperate with existing legacy (or legacy++) code. Eiffel was conceived from the start as a**component combinator**: a tool for combining software elements written in various languages, taking full advantage of the architectural capabilities of consistent object technology (classes, information hiding, Design by Contract, multiple inheritance, genericity).

The language has a provision for including software elements written in other languages (**external** clause), which can then be repackaged through Eiffel into a coherent O-O structure; ISE Eiffel also supports calling Eiffel mechanisms from the outside, through the Cecil library. The environment interoperates with many other industry-standard tools, such as OLE/COM and CORBA, and offers migration tools such as the Legacy++class wrapper for C++.

.

## How object-oriented?

| **Eiffel** | **C++** | **Java** | **Smalltalk** |
|---|---|---|---|
| Purely OO | Hybrid | Some proponents say it's pure-O-O; to us it looks pretty much like a C extension. | Purely OO |

**Our comment:** Purity is not a dogma, but an O-O language must treat the O-O paradigm seriously and without reservations if companies using it are to gain the economic benefits that they expect from object technology. Here we differ strongly from the C++/Java school; we think it is important to ensure compatibility with C (and C++) software, but that one should not pollute the language with constructs from another era.

## Age and track record

| **ISE Eiffel** | **C++** | **Java** | **Smalltalk** |
|---|---|---|---|
| Since 1985. Many reference sites, successful projects in the 800,000-line range. | Approximately 12 years old, many projects (but hard to know how many are truly O-O). | Since 1995, initially geared towards programming small appliances and Internet applets, then scaled up. | Approximately 20 years old. |

**Our comment:** Eiffel is a proven solution with many mission-critical projects to its credit. And you know that they are truly O-O, because there is nothing else in Eiffel.

Special thanks to Keith Gunn of SHL Systemhouse for providing the framework and first implementation of this comparison. The comparison is based on commonly available product literature from a broad selection of object technology vendors available at the time of release. We welcome constructive feedback and suggestions for improvement.