Eiffel programming language

Eiffel was created in 1987 by Bertrand Meyer his company Interactive Software Engineering (ISE). It closely follows Dr. Meyer’s work in Object Oriented Software Construction, Second Edition. Eiffel differs from most popular languages in several ways. Eiffel is an object-oriented (OO) programming language which emphasizes the production of robust software. Many find its syntax to be reminiscent of Pascal. Eiffel is strongly statically typed, with automatic memory management (typically implemented by garbage collection).

Eiffel is a mature OO language with a generally excellent reputation among those who are familiar with the language. However, Eiffel has failed to gain much interest from software developers. The reasons for this lack of interest are unclear, and are a topic of frequent discussion within the Eiffel community.

Distinguishing characteristics of Eiffel include Design by Contract™, liberal use of implementation inheritance including multiple inheritance, value types implemented via value semantics, and generic classes. Eiffel has a unified type system—because all types in Eiffel are classes, it is possible to create subclasses of the basic classes such as INTEGER.

Eiffel has operator overloading, including the ability to define new operators, but does not have method overloading.

The aim of Eiffel is to improve the quality of software systems and the productivity of the development process. It particularly promotes the production of software that has the following qualities: reliability (absence of bugs) extendibility (ease of change) reusability (reliance of libraries of packaged components) and portability (adaptability on many platforms with full source compatibility). It supports multiple inheritance, genericity, polymorphism, and encapsulation. Its most important contribution to software engineering is Design by Contract, in which assertions, preconditions, postconditions, and class invariants are used to assist in assuring program correctness without sacrificing efficiency. Eiffel also makes it possible to produce compilers, such as ISE’s, which generate extremely efficient code. Eiffel also offers multiple class inheritance. Many people (such as the designers of Java) have objections to multiple inheritance. The Eiffel implementation of multiple inheritance, in the opinion of its supporters, successfully meets these objections.
Eiffel's design is closely based on OOP theory, with less influence from other paradigms or support for legacy code. The language has formal support for abstract data types. In accordance with Self Documentation, a software text should be able to reproduce its design documentation from the text itself. Eiffel accomplishes this by using a formalized implementation of the Abstract Data Type.

Eiffel Studio, an integrated development environment for Eiffel, offers an object-oriented interface for software engineering. However, many programmers disliked it because it was very different from user interfaces for other integrated development environments.

Features

As a language Eiffel is a "pure" object-oriented language (arguably the most systematic application of object-oriented principles in existing languages) based on a small number of powerful concepts:

- Seamless development: Eiffel is applicable to the entire lifecycle, from analysis and high-level design to implementation and maintenance, providing a single conceptual framework throughout the software process.
- Classes, serving as the sole basis for both the module structure and the type system.
- Inheritance for classification, subtype and reuse.
- A careful and effective approach to multiple inheritance (renaming, selection, redefinition, undefinition, repeated inheritance).
- Assertions for writing correct and robust software, debugging it, and documenting it automatically.
- Disciplined exception handling to recover gracefully from abnormal cases.
- Static typing with no loopholes in the type system for safety.
- Dynamic binding for flexibility and safety.
- Genericity, constrained and unconstrained, for describing flexible container structures.
- Open architecture providing easy access to software written in other languages such as C, C++ and others.

Elegance, simplicity, or fascism?
The Eiffel language aims to promote clear and elegant coding. Eiffel emphasizes declarative statements over procedural code, and it eliminates the need for bookkeeping instructions.

More controversially, Eiffel intentionally limits stylistic expression, providing no room for clever coding tricks or coding techniques intended as optimization hints to the compiler. Some software developers feel constrained by Eiffel’s simplicity, sometimes to the extent of describing Eiffel programming as “bondage and discipline”.

**Lexical simplicity**

Eiffel is not case-sensitive. Both keywords and identifiers can be written in any combination of upper and lower case. The tokens *Make* and *make* and *MAKE* all refer to the same identifier. Coding style standards, however, generally prescribe the use of all-capitals for class names, all lower-case for variables and method names, and initial capitals for constants.

Eiffel’s syntax can be parsed without requiring end-of-statement markers. The use of semicolons as statement terminators or as statement separators is left to the discretion of the programmer. Putting a semicolon in or leaving one out will make no difference except in the unusual case of a statement starting with a left parenthesis. Most Eiffel programmers choose to omit semicolons except when putting multiple statements on a line.

Eiffel requires that sections and clauses appear in a specific order.

In contrast to most members of the curly brace family of programming languages, Eiffel does not permit expressions to be used as statements, nor statements to be used as expressions. Accordingly, a method which returns a value can only be used in expressions, while a method which does not return a value can only be invoked by method call statements.

This philosophy—that expressions and statements are fundamentally different in nature—is expanded into the concept of Command-Query Separation (CQS). Under CQS, a query method (a function which returns a value) must not change the state of the object, while a command method will change the state of the object but will not return a value. CQS is strongly recommended for Eiffel programming, but is not actually enforced by the Eiffel system.
Unlike most programming languages, Eiffel is not normally displayed in a monospaced typeface. The recommended display style is to use a proportional typeface. Keywords are displayed in bold, user-defined identifiers and constants are displayed in italics. Standard upright (roman) style is used for comments, operators, and punctuation marks.

**Syntactic simplicity**

Eiffel has only six basic executable statements:

- assignment
- object creation
- method call
- conditional
- iteration
- choice (case)

Perhaps uniquely among the OO languages, Eiffel does not permit storing into member variables of other objects. The assignment statement can only change the value of a member variable of the current object, or a local variable of the current method. All changes to other objects must be accomplished by calls to methods of that object. Direct access to member variables of other objects is “read only” in Eiffel.

The iteration (loop) statement in Eiffel is unusual in that it does not provide a field or clause which will step the loop. The programmer must explicitly code the appropriate stepping statement within the loop. For example:

```plaintext
from i := 0 until i >= 10 loop
  my_array.put (0, i)
  i := i + 1
end
```

The example above also illustrates another unusual aspect of Eiffel: there is no special syntax for accessing elements of an array. An array is simply an instance of the class `ARRAY`, and access is made through ordinary method calls. Some Eiffel compilers provide specialized optimizations for array access.

Eiffel’s procedural coding is strictly structured. There are no Eiffel statements for terminating a loop early, nor for exiting a method early.

**Non-OO operations**
Eiffel is a purely object-oriented language. Any coding which must be "close to the machine" is expected to be done in C, and Eiffel provides a straightforward interface to the C routines. Eiffel tends to be quite closely connected to C; three of the four Eiffel compilers produce C code rather than object code.

Hello World in Eiffel

class HELLO_WORLD
creation make
feature
  make is
    do
      print ("Hello, world!\n")
    end
end

The official language specification for Eiffel is Eiffel: The Language, first edition, second printing (ETL2), but that book is reportedly out of print. Bertrand Meyer is working on a new version of the Eiffel language specification, but no completion date is available (as of early 2003). In the interim, ISE has made a significant subset of ETL2 available online.

The specifications for the Eiffel language and its basic libraries are nominally under the control of the Nonprofit International Consortium for Eiffel (NICE). Although NICE has sporadically worked on improving the specifications for the basic library classes, it has done little with the language itself aside from ratifying ETL2 and a few changes which already had appeared in the available development systems. In June 2002 ISE submitted the Eiffel language to the European Computer Manufacturers Association (ECMA) for standardization, a move which effectively deposed NICE as the standards body for the language proper.