# Java review

## Abbreviations

In this review, the following abbreviations are used:

* OS: Operating System
* JVM: Java Virtual Machine

## History

Java is a general purpose programming language ( embedde languages -> 5. semntics). It was released together with the Java platform in 1996 by a team at Sun MicroSystems (now Oracle), whose leader was James Gosling. The Java language has been defined as a core element of the Java platform, until Java 7. The project started in December 1990, with the name *Stealth Project,* then renamed *Green Project*. Other relevant developers are Mike Sheridan and Patrick Naughton.

The language was initially called Oak after an oak tree that stood outside Gosling's office. Later the project went by the name Green and was finally renamed Java, from Java coffee, the coffee from Indonesia.

The original target of the project was interactive television, since the team was dreaming of combining digital consumer devices and computers, to exploit the power of the latter in the everyday life. Indeed, the Green Team demonstrated their new language with an interactive, handheld home-entertainment controller that was originally targeted at the digital cable television industry. Considering that it was just the 90’s, it really was a dream, and unfortunately the concept was much too advanced for the team at the time and didn’t succeed.  Luckily, it was successfully employed in the new born Internet. Since there weren’t languages able to provide dynamic content to HTML pages, Java filled this gap allowing, through the so-called *Java applets,* the creation of dynamic, OS independent, and browser independent content for the web. All the original success of Java comes from this.

Moreover, the main goal of the project was to offer to developers a way of achieving the principle WORA (*write once, run anywhere).* Indeed, Java platform is a software product which includes an execution engine (called a Virtual Machine), a compiler and a set of libraries. The deployment of a Java program works as follows: the compiler produces java bytecode from the original source code, then the Virtual Machine, which is different per each OS, executes the bytecode in the context of the specific machine. This allows portability of java programs, since the developers don’t have to take care of the underlying machine, and a compiled program can run everywhere without the need for recompiling it in the new machine. Java platform have been provided for almost every machine, making the Java language so popular even after nowadays.

Between November 2006 and May 2007 Sun has released almost the entire Java technology under the GNU General Public Licence, making it free.

The language evolved during time: as an example, the original applets, which had so much importance, are now deprecated. The current supported versions are Java 13, released in October 2019, and Java 11, released on September 25, 2018.

## 

Figure 1: A Java program

## Characterisation of the language

Java is (primary) an object-oriented language. A Java program is organized as a set of classes, where each class corresponds to the declaration of a new user-defined type. This is the simple way to create new types in Java: it doesn’t have explicitly type constructors. Each class allows for the declaration of variables (*attributes* in Java) and functions (*methods)* which compose the class. With attributes and methods, the user can specify characteristics and possible operations on the type itself. The instances of classes are called objects. Attribute values distinguish single objects inside a *class* type and are a representation of the *state* of an object at every execution point of the program. Methods are used to manipulate instances, such as changing their attributes’ values. All classes names should start with uppercase letters, while methods and variable names start with lowercase letters. The main program is represented by a special method of a class, called *main,* which is the portion of code executed when we run a program.

When defining a new class, the developer may specify how to create objects of that class by adata constructor*:* a constructor is simply a method which has the same name of the class, does not have a return type, and returns an instance of the class. A constructor may have parameters. The user can specify as many constructors as desired, but all with different numbers of parameters. If no constructor is provided, Java provides a default constructor, where all the attributes are set at some default values (which is null for user defined types). Qua discuti dei costruttori se hai tempo

An object of a class (i.e., an instance of a user defined type) can be created via the *new* operator, which is applied to a constructor. It allocates at runtime the necessary memory and return a reference of the new object. Java allows the user to create new objects and bind a reference of the new object to a *variable*. A variable declaration must have a unique name and a type (either primitive or user defined). Java is then *explicitly typed*: all the variable declaration must be accompanied by the type of the variable. A variable name is associated with the corresponding value in the main memory only if the variable is of a primitive type (see below), while if it’s of a class, it contains only a reference to the starting point in memory where the space for the variables of that class is allocated.

Java is also *statically typed*: all the types (of program variables, class attributes, and methods) must be known at compile time and are controlled by the type checker for consistency. If not consistent, an error is raised at compile time. java

Java enforces also *static (or lexical) scoping* of variables: the scope of a variable name is the portion of the program in which that name is associated with a particular variable. Static scoping means that the binding between a name and a variable is done at compile time and not at run time (which is dynamically scoping). If the local block of code that uses or assigns a variable doesn’t contain the declaration or initialization of that variable before the use or assign, the compiler looks at the ascendent blocks (*static parents*) as they are lexically written in the code to do the binding name-variable. If no variable declaration has been found, a compile time error is raised: contrary to JavaScript, Java requires that all variables are declared before using or assigning them.

Since Java does not support natively functional programming (see below), methods are not first-class citizens of the language and cannot be passed as arguments to other methods. Moreover, all the arguments of a method are evaluated before the method itself regardless if this is needed in the called method (*eagerly evaluation,* contrary to *lazy* *evaluation*). Then, when calling a method, Java does not pass the references of the arguments, but stores their values into new variables (*call by value*, contrary to *call by reference*). Java supports lazy evaluation only for Boolean && and || operators, which will not evaluate their right operand when the left operand is false (&&) or true (||) (*short circuit evaluation*) and the ?: operator, which evaluates a Boolean expression and subsequently evaluates only one of two alternative expressions (of compatible type) based on the Boolean expression's true/false value.

**class** **List**<E> {

E value;

List<E> next;

}

Figure 2: a simple definition of a list

Java allows for recursive data types: a class can contain an instance of the same class as attribute. As an example, see figure2, where a simple recursive list is defined. To avoid infinite recursion which would cause a segmenta, in Java variables can be *null* (since, as already mentioned, variables are references, it means that they not reference any object)

The code in figure 2 is relevant also to spot another Java feature: *polymorphism*.

Recent versions of Java are not only a simple object-oriented language but have included also some support for functional programming.

### Primitives types

Apart from user-defined types, Java provides the following already build in types (*primitives types*):

* Byte: 8-bit signed two's complement integer. It takes values between -128 and 127 (inclusive);
* Short: 16-bit signed two's complement integer. It takes values between -32,768 and 32,767 (inclusive);
* Int: 32-bit signed two's complement integer. It takes values between -231 and 231-1 (inclusive). It can be used also to represent an unsigned 32-bit integer, which has a minimum value of 0 and a maximum value of 232-1. The *Integer* class allows indeed to do that;
* Long: 64-bit signed two's complement integer. It takes values between -263 and 263-1 (inclusive). It can be used also to represent an unsigned 64-bit integer, which has a minimum value of 0 and a maximum value of 264-1. The *Integer* class allows indeed to do that;
* Float: a single-precision 32-bit IEEE 754 floating point;
* Double: a double-precision 64-bit IEEE 754 floating point;
* Boolean: only two possible values (*true* and *false*);
* Char: a single 16-bit Unicode character;

In addition to that, Java provides support for character strings, through the *String* class. String objects are immutable, which means that once created, their values cannot be changed.

Java allows for embedded languages

Java has some functional programming -> closuressss (talk about that)

Java concurrency

Global and local variables????

Subtyping!!!

Exceptionsss

## Strengths and weaknesses of the language

* Manca l’ereditarietà multipla
* Java funzionale ( che però è stato introdotto di recente)
* Sandboxing -> non danneggio l’host
* Construttori che non ci sono più
* Static scoping allows to easy checking erros
* Statically typed -> all type errors are known at compile time

## References

Notine

* Subset interpretation >> coercion interpretation ( non per i tipi primitivi però)
* Come funziona l’overloading in Java? Il tipo ritornato deve essere lo stesso)