DE GRUYTER
OPEN

10.2478/cris-2014-0006

PROGRAMMING LANGUAGE PARADIGMS & THE MAIN PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING

JAN BARTONÍČEK

This paper's goal is to briefly explain the basic theory behind programming languages and their history while taking a close look at different programming paradigms that are used today as well as describing their differences, benefits, and drawbacks.

GENERAL DEFINITION OF PROGRAMMING LANGUAGES

Programming language is so-called 'formal' language, created to make communication between the computer and its programmer easier. The very first computers were programmed using switches and plugboards, but this concept quickly evolved into software programming. To start with, programmers used machine code, which was hard to read and debug, and the invention of programming languages came to make these tasks easier.

Programming language is a set of commands, strings of characters readable by programmers but easily translatable to machine code; it has syntax, grammar, and semantics. Syntax is a set of rules that define how the commands have to be arranged to make sense and to be correctly translatable to the machine code. Grammar is a set of rules of using different punctuation, quotation marks, semicolons, and other symbols to divide and clarify the syntax of a particular language. The last component of programming language is semantics, a set of meanings assigned to every command of the language and is used to properly translate the programme to machine code.

Programming languages are often divided into three generations:

The first generation of programming languages were used to directly control the processor and were written mainly in binary or machine code. It was very hard to write the programmes and even harder to debug them.

The second generation of languages are also called low-level languages, and they use symbolic addresses and simple instructions to make programming easier and faster. These languages have access very close to the hardware itself, and they are still used to write highly-optimised code for specific hardware.

The third generation of languages use a high level of abstraction, using advanced commands, variable names, and pointers. These languages are mostly hardware independent and portable (Nasir, 1996).

BRIEF HISTORY OF PROGRAMMING LANGUAGES

The very first device to be called a computer was the Pascaline, an automatic mechanical calculator invented by Blaise Pascal in the year 1642. It was able to perform addition, but because of its complicated nature, it did not gain popularity. In 1833, Charles Babbage, a mathematician, devised plans for his "analytical engine", a steam powered machine able to carry out almost any mathematical operation. It was programmed by exchanging gears in the machine that made it extremely impractical. The design was so advanced that it was not possible to build it with the level of technology at the time. Ada Augusta King, Countess of Lovelace, called Ada Lovelace for short, was a pioneer in the computing field. She made major enhancements to the analytical machine, such as subroutines and conditional execution.

The next big advancement in computing came a hundred years later with the publication of Alan Turing's paper called "On computable numbers, with an application to the Entscheidungsproblem" in the year 1936. In it, Alan Turing proposed a theoretical machine, later named the Universal Turing Machine, which consisted of a read/write head with memory for holding instructions and an infinite tape comprised of cells able to hold one symbol each. The head was capable of moving above the tape, read its contents, alter it, and write new data to the tape. This paper introduced basic concepts of computing and its publication marks the start of a rapid development in computer technology (Armbruster *et al.*, 2001a).

Not long after the Turing's paper, the first digital computers started to be constructed. The first one to be finished was the ENIAC (Electrical Numerical Integrator and Calculator) in the year 1942. It was programmed using switches and plugboards, which was extremely impractical and time-consuming.

Interested in the success of ENIAC, a mathematician working at the Institute of Advanced Study, John von Neumann, started to work on improving the ideas behind computers, and he introduced ideas that are now collectively called the "Von Neumann architecture". He stated that computers should be static in their hardware design and their only part subject to change should be their software, making them much easier and quicker to reprogramme (Armbruster *et al.*, 2001b).

Another revolutionary woman, Grace Murray Hopper, while working on the UNIVAC, ENIAC's competitor, constructed a first compiler, a machine or later a block of code that was able to translate commands into machine code. She introduced the first programming language, called Assembly, which used slightly more readable commands in a three-address machine code ([ADD 00X 00Y 00Z] that would be used to add together X and Y and assign the output to Z).

The next major step in development of programming languages was the announcement of FORTRAN (FORMula TRANslation). It was designed by scientists at IBM and featured a revolutionary compiler with ability to optimise the compiled code. It became the very first successful high-level programming language. The first programming language that implements Object-oriented principles is Smalltalk released in the year 1970. In fact, it is purely Object-oriented, which means no data or functions can exist outside of objects (Armbruster *et al.*, 2001c).

One of the widely used languages of today is C++, which was created as an Object-oriented version of C, a very popular language invented in 1972 for development of the UNIX system. C++ inherited a wide variety of features from C, including its ability to use low-level commands to create fast and tuned programmes.

Two of the three major languages of today were both released in the year 1991, first one being Java, focusing on object-oriented approach and simplicity of the language, relying on extension libraries. The second one is called Python and it features a simple and English-like syntax, which makes it easy to use and effective. The last major language is C#, developed as a modernization to the C language. It was made public in the year 2000 as a part of the Microsoft's .NET Framework project. It is a high-level, multi-paradigm language focused on the effectiveness of program creation. (Armbruster *et al.*, 2001d)

PROGRAMMING PARADIGMS

By the word paradigm, we understand a set of patterns and practices used to achieve a certain goal. In this essay, the word 'approach' is used as a synonym to 'paradigm'. For an idea to become a paradigm, it should be picked up globally in many independent organisations and societies. There are many programming paradigms in use today, but only the three major paradigms are in this essay's discussion.

PROCEDURAL PARADIGM

Procedural paradigm is the approach to programming that was used from the beginning of computing. In this paradigm, the programme comprises of a list of instructions for the computer to execute in the order in which they were written, unless stated otherwise. It is a simple approach, and tends to be easily readable when the programme is reasonably short. Larger programmes written with a procedural paradigm in mind can be very hard to read, manage, and debug. Most procedural languages have flow control structures, IF

statements used to branch the code execution, FOR and WHILE for repeated execution, and GOTO for 'jumping' between lines of code.

When talking about a procedural paradigm, we cannot forget to mention its subset, called a structural paradigm which omits the GOTO command, meaning that the execution of the programme has to go through the entire programme without skipping any commands.

Programming languages based on procedural approach includes most of the early languages like Assembly, FORTRAN, and some more recent languages, for example, C and PHP.

Following is an example of procedural implementation of factorial function in pseudo-code, an English-like code used for language agnostic representation of programmes:

```
START
Initialize n, f = 1
Read n from the user
For i between 1 and n do:
  f = f * i
Return f
END
```

FUNCTIONAL PARADIGM

Functional paradigm is fundamentally different from the procedural paradigm in the way programmers develop their programmes. As opposed to the procedural approach, in which the programmers define the way the programme functions, in the functional approach, the programmer defines the desired outcome and cannot define the way the functions are executed. Functional programming relies on the concept of λ -calculus (lambda-calculus), which defines a way to convert any computable (as defined in Turing's paper "On computable numbers") function to a mathematical function expressible in functional languages. The most important functional programming language is LISP (LISt Processing). Some more examples of functional languages are Scheme (a dialect of LISP), Haskell, and Mathematica.

The following is an example of functional implementation of a sum function in Haskell (Computerphile, 2013):

```
sum :: [int] --> int
sum [n] = n
sum(n, ns) = n + sum ns
```

OBJECT-ORIENTED PARADIGM

The Object-oriented paradigm was developed by Alan Kay while he was working on Smalltalk. It was developed to make large projects easier to manage and share the workload among coworkers and is simplified thanks to the modularity of objects.

The main focus of the Object-oriented paradigm is an object. Object is a way of grouping data structures and functions that can be carried out on the data, which are called methods (Oracle, 2013a). Using the proper typing of the data and methods, the programmer can achieve a high level of encapsulation. For further description of object, see the graphic below.

Object-oriented programming is based on four main concepts:

THE CONCEPT OF CLASS

Class is a template from which objects are created. It contains the description of all the data structures and all methods of the resulting objects. Most classes have their constructors, methods, which initialise the created objects' variables depending on the data given.

THE CONCEPT OF INHERITANCE

Inheritance is the ability of classes to inherit data structures and methods from other classes (Oracle, 2013b). For example, we can imagine a parent class vehicle and its child classes: Car, Bicycle, and Motorcycle. For further description of inheritance, see the graphic below:

An object created from the class Bicycle shown in the graphic will have not only the methods and variables declared in the Bicycle class, but all the methods and variables inherited from the Vehicle class. The complete class Bicycle without the use of inheritance is shown in the graphic below.

```
Class Bicycle

method Drive(), method Steer(), method Brake(), method Pedal(),
method ChangeGear()

variable numberOfWheels = 2
```

THE CONCEPT OF ENCAPSULATION

Encapsulation is a property of a well-written object. The programming language provides the programmer with tools to prevent other objects or programmes to access the data held within the object directly, allowing him to specify the ways the data can be manipulated using public methods, which are used to validate input data before storing it.

Closely related to encapsulation is the notion of modularity. Modularity of objects, the isolation of the working code from the outside world, is used to make larger programs easier to develop and manage. The programmer does not need to understand how a certain object works, he only needs to know which methods to use for accomplishing his goals. For example, the programmer does not understand how the engine of a Motorcycle from the previous example works, but he knows that he needs to call the method ThrottleUp() in order to increase the speed.

THE CONCEPT OF POLYMORPHISM

Polymorphism is the ability of objects to process multiple data types in the same object. This ability is based on one of the notions used in most programming paradigms called function overloading, which allows the programmer to specify different functions under the same name, whose execution will be dependent only on the data type of a given argument. For example, the programmer would write two different functions under the name Print() for outputting information to the screen. First one would be defined as Print(int input) and it would handle printing integers and the second one would be defined as Print(string input) and it would handle printing strings of characters. This allows the programmer to use the function Print() to print integers and strings of characters instead of using two different functions PrintInt() and PrintString().

OBJECT-ORIENTED LANGUAGES

The first programming language to implement features of the Object-oriented paradigm was Smalltalk. One of the main contributors to its development, Alan Kay, is the creator of the Object-oriented paradigm. It was released in the year 1970. It is purely object based, which means that no data or methods can exist outside of an object. The next programming language focused on the Object-oriented paradigm was a set of extensions for C collectively called 'C with Classes', which later led to the development of one of the most influential Object-oriented language of today, C++.

Another well-known Object-oriented language is Java, widely used today for creation of programmes ranging from servlets for interactive websites to mobile and computer games. It focuses on cross-platform compatibility using Java Virtual Machine to run the same code on a wide range of different types and architectures of hardware. Although practical for portability, this approach makes Java programmes less effective in resource usage and slower than other programming languages.

CONCLUSION

Although it is not without its quirks, the Object-Oriented paradigm is the most frequently used paradigm of today. It owes its popularity to the simplicity of large project management using object modularity and encapsulation. Every programming paradigm has its own advantages and disadvantages for many usage scenarios and that is why most of the currently used programming languages support multi-paradigm approach, which allows the programmer to choose the best development practice for his specific needs. ■

REFERENCES

- Armbruster, F., Brooks, C., Christos, B., Fuller, K. (2001a) *The Birth of Modern Computing and Programming Ideas*. Available at: <http://www2.lv.psu.lv.edu/ojj/ist-240/reports/spring2001/fa-cb-bc-kf/1200-1940.html> (Accessed: 12 October 2013).
- Armbruster, F., Brooks, C., Christos, B., Fuller, K. (2001b) *The Concepts become reality*. Available at: <http://www2.lv.psu.lv.edu/ojj/ist-240/reports/spring2001/fa-cb-bc-kf/1941-1950.html> (Accessed: 12 October 2013).
- Armbruster, F., Brooks, C., Christos, B., Fuller, K. (2001c) *Programming as we know it begins*. Available at: <http://www2.lv.psu.lv.edu/ojj/ist-240/reports/spring2001/fa-cb-bc-kf/1951-1970.html> (Accessed: 12 October 2013).
- Armbruster, F., Brooks, C., Christos, B., Fuller, K. (2001d) *The Arrival of the Internet Languages*. Available at: <http://www2.lv.psu.lv.edu/ojj/ist-240/reports/spring2001/fa-cb-bc-kf/1991-2001.html> (Accessed: 12 October 2013).
- Computerphile (2013) *Programming Paradigms – Computerphile*. Available at: <http://www.youtube.com/watch?v=sqV3pL5x8PI> (Accessed: 21 October 2013).
- Nasir, M. J. B. M. (1996), *Generations of Programming Languages*. Available at: http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol2/mjbn/article2.html (Accessed: 15 October 2013).
- Oracle (2013a) *What Is an Object?* Available at: <http://docs.oracle.com/javase/tutorial/java/concepts/object.html> (Accessed: 10 October 2013).
- Oracle (2013b) *What Is Inheritance?* Available at: <http://docs.oracle.com/javase/tutorial/java/concepts/inheritance.html> (Accessed: 10 October 2013).