Comparison of two object-oriented languages: C++ and Ruby

Jeroen Heymans

December 26, 2011

1 Introduction

1.1 Foreword

This paper is written for the master course Principles of Object Oriented Languages on the VUB. It contains a comparison between two languages that contain object-oriented features.

1.2 The compared languages

The choice for this paper was C++, a well-known static typed language with good object oriented features, and Ruby, a dynamic typed language that was designed to be as object-oriented as possible.

1.2.1 C++

Originally called "C with classes", C++ is a multi-paradigm language based on the popular language C. C++ doesn't require you to write all code in an object-oriented fashion, it is also possible to write all your code in functions.

1.2.2 Ruby

The creator of Ruby, Ykihiro Matsumoto, looked at other languages to find an ideal syntax. He watned a scripting language that was more powerful than Perl and more object-oriented than Python. In Ruby, everything is an object. It was influenced by Smalltalk, every type was given methods instance variables.

1.3 What follows next

We will compare several important features that belong to object-oriented languages like inheritance, polymorphism, access control and many more.

2 Comparisons

2.1 Classes

In object-oriented programming, a class is a blueprint that is used to construct objects which we call object instances. We compare the several types of classes that can be made.

2.1.1 Abstract classes

An abstract class is a class that can not be instantiated for it is either labelled as abstract or it specifies abstract methods. The general idea of abstract classes is that the class must be extended. In C++, an abstract class is a class having at least one pure virtual function. For example, this is an abstract class in C++:

```
class A {
  public:
    virtual void f() = 0;
};
```

Abstract classes in Ruby are however a different story. Unlike some object-oriented languages, Ruby does not support a keyword "abstract" or a way like C++ to define an abstract class. However, the programmer himself can force a class to be extended when he makes *:new* private. The extending class must then make *:new* public:

```
class A
  private_class_method :new
  def initialize(txt = "Hello!")
    puts txt
  end
end

class B < A
  public_class_method :new
end
```

2.1.2 Concrete classes

2.1.3 Inner classes

2.1.4 Metaclasses

In Ruby, a class is also an object. Each class is an instance of the unique metaclass which is built in the language.

2.1.5 Non-subclassable

2.1.6 Partial classes

2.1.7 Uninstantiable classes

2.2 Inheritance

3 Conclusion

4 References

http://www.ruby-lang.org/en/about/
http://www.cplusplus.com/info/history/
http://publib.boulder.ibm.com/infocenter/lnxpcomp/v8v101/index.jsp?topic=http://www.klankboomklang.com/2007/1
metaclass/