

C++ versus Ruby

Jeroen Heymans
Rolnr. 90055
jeheyman@vub.ac.be

The compared languages

- C++
 - Static typing
 - Multi-paradigm
- Ruby
 - Dynamic typing
 - Duck typing
 - Object oriented

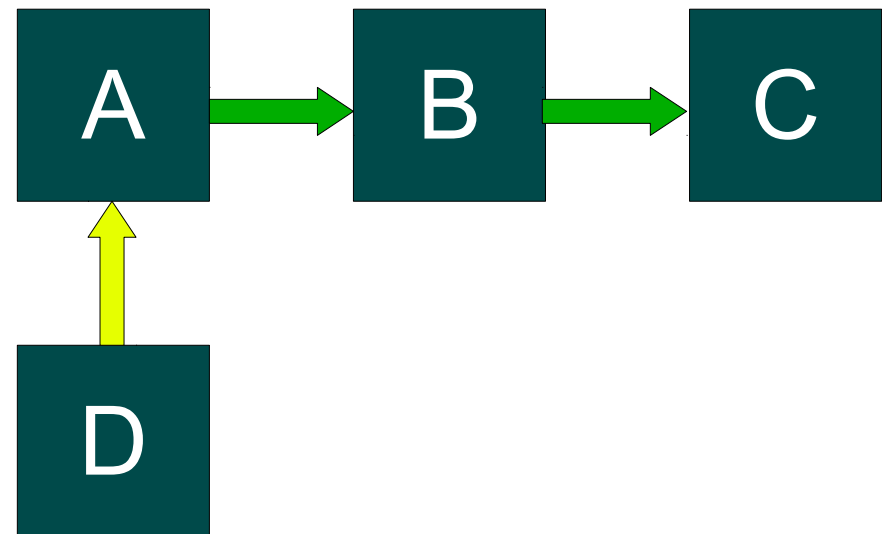
Abstract class

- C++
 - Virtual method
 - All virtuals?
- Ruby
 - Private constructor

```
1 class A // abstract class
2 {
3     public:
4         virtual void print()=0;
5 };
6
7 class B // not an abstract class
8 {
9     public:
10         void print() {
11             std::cout << "Hello!";
12         }
13 };
```

Friend class

- Extra access privilege
 - A is friend of B? A gets access to everything from B
- Only in C++
- Special Ruby libraries
- Friendships
 - Not corresponded
 - Not transitive
 - Not inherited



Inner class

- Inner class defined in outer

- C++

- Keep instance to inner and outer class
- Inner class can access privates from outer class

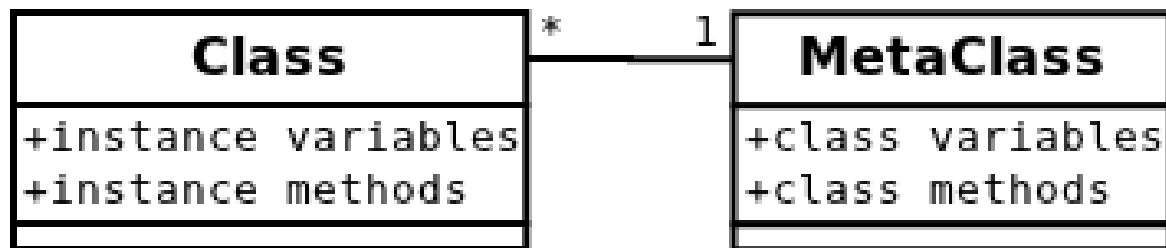
- Ruby

- Keep instance to inner and outer class
- Act is if they are separate classes

```
1 class A {  
2     private:  
3         int a;  
4     public:  
5         A(int newA): a(newA), b(this) { }  
6         class B {  
7             public:  
8                 B(A* a): outer(a), test(1) { }  
9                 int f() {  
10                     return outer->a;  
11                 }  
12                 A* outer;  
13             public:  
14                 int test;  
15         } b;  
16         int f() {  
17             return b.test;  
18         }  
19 };
```

Meta class

- Only in Ruby
- 1 class has 1 meta class
- Class methods versus instance methods
- Class members versus instance members



Partial classes

- C++
 - Only splitting codefiles
- Ruby
 - Dynamic addition of members and methods
 - Open classes
 - Can be *frozen*

```
1 class String
2     def doublePrint
3         print self
4         print self
5     end
6 end
7 puts "test".doublePrint
```

Access control

- Public, protected, private
- Difference in private
 - C++
 - Only accessible via class
 - Other objects of same class
 - Ruby
 - Subclasses can access privates
 - Implicit versus explicit receiver

```
1 class A {  
2     private:  
3         void foo() { return; }  
4     public:  
5         void test(A* a) {  
6             foo();  
7             this->foo();  
8             a->foo();  
9         }  
10 };
```

```
1 class A  
2     private  
3         def foo  
4         end  
5     public  
6         def test(obj)  
7             foo  
8             self.foo  
9             obj.foo  
10        end  
11 end
```


Access control derived classes

- C++ feature
- Derive as:
 - Public
 - Public and protected stay public and protected
 - Protected
 - Public and protected become protected
 - Private
 - Public and protected become private

```
1 class A {  
2     public:  
3         void f() { std::cout << "Wazaaa"; }  
4 };  
5  
6 class B : public A {  
7     public:  
8         void f() { A::f(); }  
9 };  
10  
11 class C: private A { };
```

Inheritance

- C++
 - Multiple
 - Explicit super
- Ruby
 - Single
 - Mixins
 - Modules
 - Include

```
1 module Animals
2   def foo
3     puts "Module A"
4   end
5 end
6
7 module B
8   def foo
9     puts "Module B"
10  end
11 end
12
13 class C
14   include A
15   include B
16 end
17
18 c = C.new
19 c.foo
```

Polymorphism

- Different response to same message
- Static typing versus duck typing

```
1 class Animal
2   def makeNoise
3     throw NotImplementedError.new("makeNoise() not implemented")
4   end
5 end
6
7 class Dog < Animal
8   def makeNoise
9     puts "Woof!"
10  end
11 end
12
13 class Cat < Animal
14   def makeNoise
15     puts "Meow!"
16   end
17 end
18
19 class Car
20   def makeNoise
21     puts "Vroom!"
22   end
23 end
24
25 def print(animal)
26   animal.makeNoise
27 end
28
29 dog = Dog.new
30 cat = Cat.new
31 car = Car.new
32 print(dog)
33 print(cat)
34 print(car)
```

Namespaces or modules

- C++: namespaces
 - Hierarchical
- Ruby: modules
 - Modules in classes possible

```
1 namespace A
2 {
3     const int test = 1;
4     void print() {
5         std::cout << "Hello!";
6     }
7     class Foo { };
8     namespace B
9     {
10         class Foo {
11             public:
12                 void test() {
13                     std::cout << "Test";
14                 }
15             };
16         }
17 }
```

Reflection

- Only in Ruby
- Instance variables
 - Get, set, remove
- Class variables
 - Get, set, remove
- Methods
 - Define, undefine, alias

Everything is an object?

- Ruby

```
5.times { print "test".length }
```

- C++
 - Primitives not

Conclusion

- Ruby
 - Lots of OO features possible
 - Extra OO features can be implemented
- C++
 - Good basis for OO development
 - Less dynamic