C++ versus Ruby

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

The compared languages

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

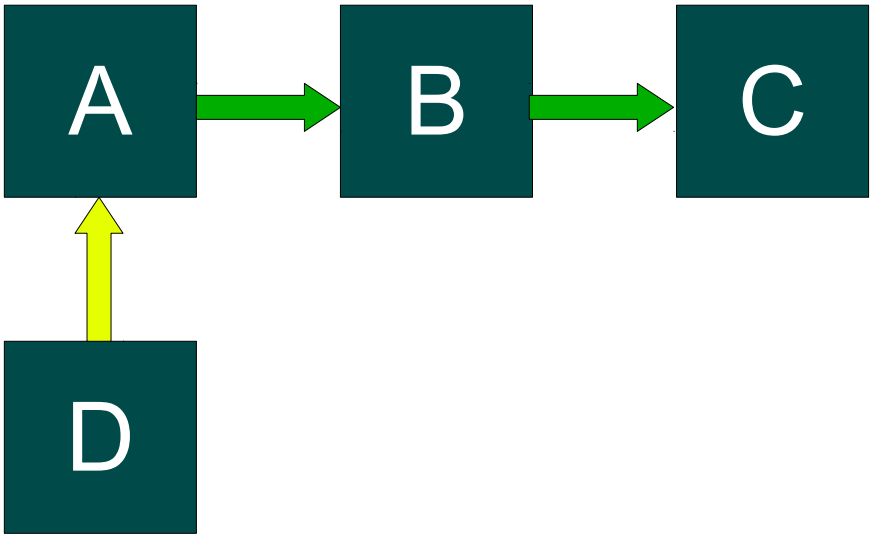
Abstract class

- 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



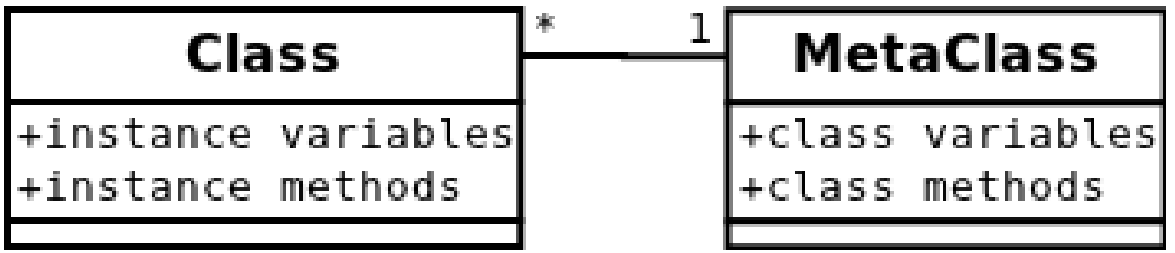
Inner class

- 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*

```ruby
class String
    def doublePrint
        print self
        print self
    end
end
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

```ruby
module A
    def foo
        puts "Module A"
    end
end

module B
    def foo
        puts "Module B"
    end
end

class C
    include A
    include B
end

c = C.new
c.foo
```

Polymorphism

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

```ruby
class Animal
    def makeNoise
        throw NotImplementedError.new("makeNoise() not implemented")
    end
end

class Dog < Animal
    def makeNoise
        puts "Woof!"
    end
end

class Cat < Animal
    def makeNoise
        puts "Meow!"
    end
end

class Car
    def makeNoise
        puts "Vroom!"
    end
end

def print(animal)
    animal.makeNoise
end

dog = Dog.new
cat = Cat.new
car = Car.new
print(dog)
print(cat)
print(car)
```

```
void print(Animal* a) {
    a->makeNoise();
}
```

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

```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