CSc 372

Comparative Programming Languages

32 : Ruby — Types

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

Compile-time type-checking

- Some call it *static checking*, *type safety*, *strict type-checking*, *strong typing*,. . .
- It does have some advantages:
  1. You catch certain errors at compile time which you now can be sure won't occur at run-time: arithmetic between the wrong types, wrong number of arguments to functions, etc.
  2. Simple errors that appear during code refactoring are easily caught and fixed.
  3. The more the compiler knows about your code, the better optimized code it can produce.
  4. Types serve as comments to the programmer, reminding him/her of what types of arguments a method was designed to take.

Compile-time type-checking. . .

- But:
  1. Even Java has many errors which cannot be caught until run-time, such as `ClassCastException` and `ArrayBoundsException`.
  2. Sometimes you need more flexibility, and it can be hard to work around a strict typechecker.

Run-time type-checking

- On the one hand, on the other hand:
  1. Less static type-checking may make programs faster to write, but it may also make them harder to maintain.
  2. A program is written once, but read and re-written many times — types can help someone unfamliar with the code to understand it quicker.

Ruby Typing

- The type of an object is defined by what it can do.
- *If an object walks like a duck, and talks like a duck, let's treat it like it's a duck!*
- We call this *Duck Typing*.

Ruby Typing

- Here's a simple class that logs data by appending it to a file:

```
class Logger
   def initialize()
      @f = File.open("logfile", "w")
   end

   def log(message)
      @f << message
   end
end

l = Logger.new
l.log("Ducks ahoy!\n")
```

Ruby Typing. . .

- Or a string, which also knows the << message.
- Notice that the only change we had to make was to the statement that creates the `f`-object.

```
class Logger
   def initialize()
      @f = ""
   end

   def log(message)
      @f << message
   end
end
```

Ruby Typing. . .

- Or an array, which also responds to the << message:

```
class Logger
   def initialize()
      @f = []
   end

   def log(message)
      @f << message
   end
end

l = Logger.new
l.log("Ducks ahoy!\n")
```

Ruby Type “Checking”

- If you absolutely want to check types, you should really check whether an object responds to a particular message or not:

```
class Logger
   def initialize()
      @f = {}
   end
   def log(message)
      unless @f.respond_to?(:<<)
         fail TypeError.new("log needs <<")
      end
      @f << message
   end
end
```

Ruby Type “Checking”. . .

- Of course, all we’re checking here is that there’s a method by the name of <<, we know nothing about what arguments it takes, what it does to those arguments, etc, so this is pretty weak checking.

Ducks vs. Dragons

```
class Duck
  def quack() puts "Quack!" end
  def walk() puts "Do the duck walk!" end
end

def playInMyPond!(someSortOfDuck)
   someSortOfDuck.quack()
   someSortOfDuck.walk()
end

donald = Duck.new()
playInMyPond!(donald)
```

Ducks vs. Dragons

```
class Dragon
  def quack() puts "Impersonate a Duck!" end
  def walk() puts "Breath fire!" end
end

def playInMyPond!(someSortOfDuck)
   someSortOfDuck.quack()
   someSortOfDuck.walk()
end

dragon = Dragon.new()
playInMyPond!(dragon)
```

Cowboys vs. Squares — Ruby

```
class Cowboy
   def move() end
   def draw() end
end

class Square
   def move() end
   def draw() end
end

johnWayne = Cowboy.new()
smallSquare = Square.new()
johnWayne = smallSquare
```

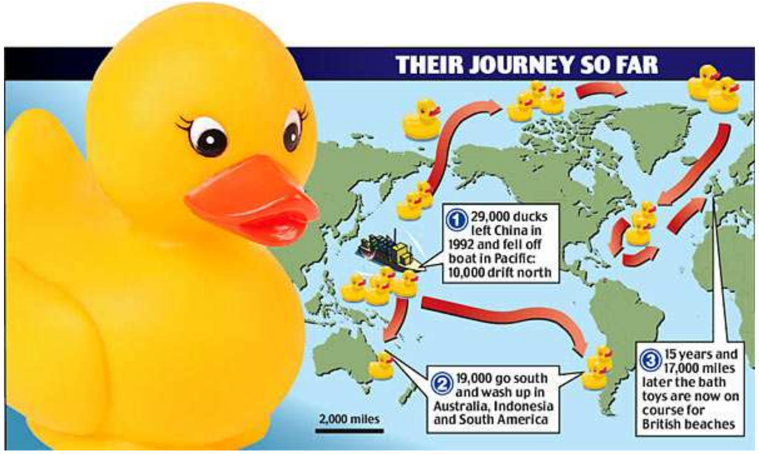
Cowboys vs. Squares — Java

```
class Cowboy {
   void move() {}
   void draw() {}
}
class Square {
   void move() {}
   void draw() {}
}
class Java {
   public static void main(String[] args) {
      Cowboy johnWayne = new Cowboy();
      Square smallSquare = new Square();
      johnWayne = smallSquare;
   }
}
```

Readings

- Read Chapter 23, page 365–377, in *Programming Ruby — The Pragmatic Programmers Guide*, by Dave Thomas.

Well-Travelled Ducks


THEIR JOURNEY SO FAR
1 29,000 ducks left China in 1992 and fell off boat in Pacific: 10,000 drift north
2 19,000 go south and wash up in Australia, Indonesia and South America
3 15 years and 17,000 miles later the bath toys are now on course for British beaches
2,000 miles

From http://www.dailymail.co.uk/pages/live/articles/news/news.html?in_article_id=464768