# The Ruby Programming Language for Java Programmers

The Ruby language was created by Yukihiro Matsumoto in 1993 and released for general use in 1995. It is popular in Japan, where it was written. But it is also used elsewhere; particularly with the Rails web programming framework know as Ruby on Rails. It is an interpreted language; unlike Java, which is first compiled to bytecode and then interpreted.

Ruby is open source and available from <http://rubyforge.org>. The most recent stable version is 1.92. It can be downloaded for Windows at <http://rubyforge.org/frs/?group_id=167&release_id=45627> . Its name either comes from the birthstone of one of Matsumoto’s friends or as a reference to other web languages such as PHP and Pe[a]rl .

## Object Orientation

Ruby is a fully object oriented language. Everything is an object inherited from *Object*. There are no primitives such as *int* or *char*. Inheritance works about the same as in Java. If the class Student is a child of the class Person, Java writes this as

**class Student extends Person**,

 while Ruby writes

**class Student < Person.**

If name and age are instance variables in Person, they will be inherited by Student without further declaration. And the keyword, *super*, is used the same in both languages.

Ruby also has *modules* that can b*e* used to group classes. The notation here is a double colon. An example from Rails is

 **class Person < ActiveRecord::Base**

Here Person is a child of ActiveRecord. And ActiveRecord is in the Base module. Modules work much the same as namespaces in other languages, such as XML.

## Variables and Symbols

Ruby is weakly typed. Variables receive their types when something is assigned to them. Java is strongly typed; instance variables are declared at top level in the class. Ruby instance variables are denoted by an @ sign, such as @name or @age. Their scope is the entire class, just as in Java, and they can be introduced anywhere in the class. Global variables in Ruby have a double @@ sign. Neither language uses these very often. There is no boolean type, but everything is *true* except for *nil* and *false*.

Access controls: *public, protected* and *private,* work prettymuch the same as in Java. When placed in a class, everything following has the given control until another control is listed.

Variables are all accessed by reference. That means that if @name is a string, @name itself contains the address of a location in memory that holds the string. Since this can involve a lot of memory locations and references, Ruby has something called a *symbol*. Symbols have a name (string) and number (integer) but no location. (They probably reside only in the symbol table for the program.) They are particularly useful as the keys in hashes and are very widely used in Rails.

## Blocks

In Java, blocks are enclosed by curly braces. You can do the same in Ruby, but more often blocks are started with some control statement and terminated by the keyword, *end*. Statements in Java are separated by semi-colons. This is done in Ruby only when several statements are on the same line. In this case, curly braces are used to determine the block. When statements are on separate lines, indentation is used to make it easy to see where to place the *ends*. The standard among Ruby programmers is two spaces.

Variables defined within a block are local to the block and have no value outside it. The exceptions, of course, are instance variables preceded with the @ sign. Blocks are often used with the keyword, *each*. This is an iterator. We will see an example later on.

Methods in Ruby begin with the keyword, *def*, and are terminated with an *end*. The body of the method is a block. If a method has parameters, they follow the method’s name and are enclosed in parentheses. Parentheses are optional for methods without parameters.

## Example Programs

The following are two programs that do the same thing, the one on the left is in Java and the one on the right is in Ruby. The Ruby program is interpreted and can be run in a special console window. The Java program must first be compiled to bytecode before it can be executed.

Java

public class People

{ public static void main (String [] args)

 { Person girl = new Person ("Alice", 5);

 girl.show\_person ();

 }

} // People

class Person

{ String name;

 int age;

 Person (String name, int age)

 { this.name = name;

 this.age = age;

 }

 protected void show\_person ()

 { System.*out*.println (name);

 System.*out*.println (age);

 }

} // Person

Ruby

class Person

 attr\_accessor :name, :age

# initialize is the same as a constructor

def initialize (name, age)

 @name = name

 @age = age

 end

# puts is the same as println

# print is the same as print

def show\_person

 puts @name

 puts @age

 end

end

girl = Person.new("Alice", 5)

girl.show\_person

## Instantiation and Initialization

Some differences are easy to see. Instead of Java’s

Person girl = new Person ("Alice", 5);

Ruby has

girl = Person.new("Alice", 5)

and the constructor in Java is an initialize method in Ruby.

Java requires that each variable be declared with a type, i.e. String name; Ruby only assigns a type when a value is assigned to the variable. Ruby instance variables begin with the @ sign, as in @name. Also Java comments begin with a double slash //, while Ruby comments begin with #. Ruby comments, like the double slash comments in Java only extend to the end of the line. For display, *puts* in Ruby is the same as *println* in Java, however *print* works the same in both languages.

The line in the Ruby program

attr\_accessor :name, :age

is a shortcut that creates *accessor* and *mutator* (getters and setters) methods for the variables @name and @age. This means that we can have statements such as

 girl.name = "Betty"

 girl.age = 7

With this, the girl’s name changes to Betty and age to 7.

## Interactivity

The Java People class is a simple program that instantiates a class called Person and then executes its show\_person method. Since Ruby is interactive, the Person class is executed in the *irb* (Interactive Ruby) console window as shown below.



Java programs must first be compiled before they can run. This saves time after the program is deployed, but an interactive language is easier to debug.

## Data Structures

Ruby has the usual data structures, including arrays and hashes. Arrays are indexed by integers beginning with zero, as in Java. But unlike Java, the contents of a location in an array may be any type. They do not all have to be the same. Hashes are really collections of key-value pairs. The keys are almost always symbols (denoted by the leading colon.) They are used more often than arrays.

An array may be assigned a list using square brackets:

 order = [“blue”, 6, 24.95]

Or you can read elements into it from a file or database. In this case you will have to create a new instance of the array before using it: order = Array.new.

Hashes are very similar to arrays, except that the index is a key and not an integer. When creating a new hash, use curly braces instead of square brackets.

 order = {:color => “blue”, :size => 6, :price => 24.95}

The key may be a string, but it is almost always a symbol. In Ruby on Rails, hashes are used to send parameters from a web page to the server. An alternate way to define a hash is with key:value, as in

 order = {color: “blue”, size: 6, price: 24.95}

To extract something from a hash, just provide the name of the hash followed by square brackets containing the key.

 order[:color], order[:size], order[:price]

## Control Structures

Ruby has all the usual loops and conditionals as well as some additional forms using the iterator, *each*, in conjunction with a block.

The conditionals take the following form:

 if order[:color] == “blue”

 …

elsif order[:size] == 6

 …

else

 …

end

Comparison operators are the same as in C and Java.

We also have *for*, *while* and *until* statements, similar to those in Java.

 sum = 0

 count = 1

while count < 10

 sum += count

 end

 puts sum

However, more often in Ruby iteration is done with the iterator, *each*.

sum = 0

[1..10].each do |count|

 sum += count

end

puts sum

Here count is a parameter to the block and has no value outside of it.

Iterating through an array or hash is common.

 for item in order do

 puts item

The keyword, *do*, is used in both of these control structures. It indicates what operation to do.

## Exceptions

Ruby exceptions work very much like those in Java. They are subclasses of the Exception class and have the following form:

 begin

 …

 rescue

 …

 rescue

 …

 ensure

 …

 end

Here *rescue* and *ensure* are the same as *catch* and *finally*. There is also a *catch/throw* combination that ‘throws’ an exception up the stack to a corresponding ‘catch’.

## Some Conventions

Ruby class names begin with upper case letters and use *camel* case, while method and variable names begin with lower case letters and separate words with underscores. Camel case (or bumpy case) capitalizes each new word in the name. As in Java, spaces are not allowed in names.

 class ActiveRecord

 def show\_person

 @little\_girl

You don’t need to follow these conventions, but it seems reasonable to do so. Other Ruby programmers will be more comfortable with your code if you do.

## References

Dave Thomas, *Programming Ruby 1.9, the Pragmatic Progammers’ Guide*, 3rd edition, The Pragmatic Programmers, 2009

Sam Ruby, Dave Thomas and David Heinemeier Hannson, *Agile Web Development with Rails,* 4th edition, 2010, Chapter 4

Steve Litt, *Ruby Basic Tutorial,* [*http://www.troubleshooters.com/codecorn/ruby/basictutorial.htm#\_About*](http://www.troubleshooters.com/codecorn/ruby/basictutorial.htm#_About), 2005

Steve Litt, *Ruby the Right Way,* [*http://www.troubleshooters.com/codecorn/ruby/rubyrightway.htm*](http://www.troubleshooters.com/codecorn/ruby/rubyrightway.htm)*,* 2005