SIMPLE CONSOLE APPLICATION USING RUBY

Ruby

A Programmer's Best Friend

Oct-2011

Compile No More!

By: William Surya Permana.

TABLE OF CONTENT

CHAPTER 3:
CREATE AN ANGRY BOSS PROGRAM

In this part of the training, you will learn how to make a program (as the boss) that ask the user (as the employee) what is his/her name, ask what he/she want, and the fired him/her.

Look at this example:

> **Okay what's your name?**
> **> Matz**
> **What do you want now, Matz?**
> **> I want a raise.**
> **WHADDAYA MEAN "I WANT A RAISE."?!? YOU'RE FIRED!!**

5.1 Output

Now, before we begin, let's learn about output in Ruby.

To print a text into the console screen, you can use these command.

- puts → print text and end it with line break.
- print → print text without a line break.
- printf → print text with specific format. To learn more about this format, see http://sharkysoft.com/archive/printf/docs/javadocs/lava/clib/stdio/doc-files/specification.htm.

You can use << to concatenated the text. For puts and print, you can these methods to modify how the program shows the text:

- ljust width → create width times spaces, and print it in the left
- rjust width → create width times spaces, and print it in the right
- center width → create width times spaces, and print it in the center

For example, if you code "puts "Ruby".center 80", it will print "Ruby" in the center of the console screen. The program will print 80 spaces and overwrite the "Ruby" text in the center. Note: you can use single quote or double quote to set the text.

You can also change the case with these methods:

- upcase → convert the text to uppercase
- downcase → convert the text to lowercase
- swapcase → invert the case, the caps will become lowercase, and the opposite.
- capitalize → capitalize the first letter and lowercase the others

5.2 Variable

To make a variable just write any plain, lowercase word. Variables may consist of letters, digits and underscores. With variables, you give a nickname to something you use frequently, or you can set it with value from user input.

Example: `pencil_price = 1000`

5.3 Input

After you learned output, let's go on to how to make a user input in Ruby.

To scan an input, you can use gets command which get the input in string. But it will also retrieve the line break. So, maybe you will need the Chomp method which will remove the line break.

Example: `inputted_name = gets.chomp`

5.4 The Case

So, with that information you can make the program mentioned in the first part. If you are still don't know it, here's the code:

```
1 puts "Okay what's your name?"
2 print "> "
3 name = gets.chomp
4
5 puts "What do you want now, " << name << "?"
6 print "> "
7 want = gets.chomp
8
9 puts 'WHADDAYA MEAN "' << want.upcase << '"?!?  YOU' << "'RE FIRED!!"
```

The first line of code will print “Okay what’s your name?” in the screen with line break.

The second will print a greater than sign and a space, without line break.

The third will take value from what the user inputted, chomped it, and set it to variable name.

The fifth will print “What do you want now, ” followed by variable name, a question mark, and a line break.

The sixth will print a greater than sign and a space, without line break.

The seventh will take value from what the user inputted, chomped it, and set it to variable want.

The last will print “WHADDAYA MEAN "” followed by variable want in uppercase, and then print “?!? YOU'RE FIRED!!”.

This is the end of chapter three.

CHAPTER 4:
CREATE AN INTERACTIVE MENU

In this part of the training, you will learn how to make a program with main menu. And the user will have to choose want he/she want. After selecting menu one and input his/her name, it will back to the main menu. But if the user select menu two, it will print a greeting and end the program.

Look at this example:

```
1. Input a name
2. Exit
> 1
What's your name?
> Matz
1. Input a name
2. Exit
> 1
What's your name?
> John Doe
1. Input a name
2. Exit
> 2
Thank you for using this program
```

4.1 Looping

Now, before we begin, let's learn about looping in Ruby.

In Ruby, there are many ways to make a loop. There are times, for, each, while, and until. Times is the simplest one to do loop. For example, if you code "2.times {puts "Hello"}", Ruby will understand it, and puts word Hello on the screen two times.

In this case, we will use until. The format is type “begin”, and followed by the statement(s) you want to loop, after that type “end until” followed by the condition.

For more information see http://www.tutorialspoint.com/ruby/ruby_loops.htm, or the answer code in section 4.3.

4.2 Selection

Now, let’s learn about selection in Ruby.

In Ruby, there are many ways to make a loop. There are if, case, and ternary operator. If is the simplest one to make a selection. For example, if you code “puts "Your number is odd." if number%2==1”, Ruby will understand it, and only puts word “Your number is odd.” on the screen if the variable number modulus by two is one, which means an odd number.

In this case, we will use case. The format is type “case”, and followed by the condition and statement(s) you want to do, after that type “end”. Use “when” to specify the condition, then you can follow it with the statement(s) in the next line(s).

For more information see http://www.tutorialspoint.com/ruby/ruby_if_else.htm, or the answer code in section 4.3.

4.3 The Case

So, here’s the code:

```
1   begin
2     puts "1. Input a name"
3     puts "2. Exit"
4     print "> "
5     input = gets.to_i
6
7     case input
8     when 1
9       puts "What's your name?"
10      print "> "
```

```
11      gets
12    when 2
13      puts "Thank you for using this program."
14    else
15      puts "Invalid input."
16    end
17  end until input == 2
```

The first and the last line of codes will mark the loop. The program will do the loop until the value of variable input is 2. Inside the loop it will print “1. Input a name”, “2. Exit”, and “> ” (line 2-4). Later, in line 5, it will scan what the user inputted, converted it to integer (using to_i), and save it to variable input.

You can use to_i to convert the scanned input into integer, to_s to convert the scanned input into string, or to_f to convert the scanned input into float.

The next part it (line 7-16) will do a selection to the variable input. When variable input value is 1, it will print “What's your name?”, and “> ” (line 9-10). Later, in line 11, it will scan what the user inputted. But when variable input value is 2, it will print “Thank you for using this program.” Else, it will print “Invalid input.”.

This is the end of chapter four.

CHAPTER 5:
CREATE A FILE WRITER

In this part of the training, you will learn how to make a program which can write a file.

Look at this example:

```
Type your document file and location (e.g. "D:\text.txt") below!
> D:\text.txt
Type your document now. Type "::end::" to finish typing:
> Hello
> World!
> I'm creating this file using my own program.
> ::end::
File saved.

Type a new document again? [Yes/No]
> AAA
Invalid input.
Type a new document again? [Yes/No]
> No
```

5.1 Writing file

Now, let's learn about file writing in Ruby.

To write a file in Ruby, you can use new method in File class (File.new), and specify the access right mode, "w" in this case. "w" mode will write a new file or overwrite existing file. If you want to append something to the file, you can use "a" mode. To write to the file, you still can use puts, print, and printf. After that, don't forget to close the file.

For example, you can code:

```
file = File.new("D:\\text.txt", "w")
file.puts("Hello World!")
file.close()
```

In that example, you will create a new file name text.txt in drive D and you can write it. The program will write "Hello World!" to the file, and finally close the file.

For more information, see http://www.tutorialspoint.com/ruby/ruby_input_output.htm.

5.2 The Case

In this case, you will need main looping, which ended when user don't want to type a new document again. You must validate them using selection. Next, in the main loop, you will need another loop that will continue append the file until user type "::end::". Now, try to make the program by yourself!

This is the end of chapter five.

CHAPTER 6:
CREATE A PRODUCT LIST MANAGER

In this last part of the training, you will learn how to make a product list manager program which can read and write a file, and also use all of the syntax you learn before.

Look at this example:

```
Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 1
PRODUCT LIST
------------------
No      Product name        Price         Stock
1.      Book                Rp. 3000      10
2.      Pencil              Rp. 1000       9
3.      Eraser              Rp.  500       5
Press Enter to continue.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 2
Name of the product:
> Pen
```

```
The price of Pen in rupiahs:
> 2500
The amount of Pen you own:
> 20
Product added successfully.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 3
Type the product name that you want to delete:
> Book
Product deleted successfully.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 4
Thank you. See you later.
```

6.1 Reading file

Now, let's learn about file reading in Ruby.

Same as writing file, to read a file in Ruby, you can use new method in File class (File.new), and specify the access right mode, "r" in this case. But, when you want to get the value of the line you can't use 'gets', you need using 'each' instead.

For example, you can code:

```
file = File.new("D:\\text.txt", "r")
file.each {|thisline|
  puts thisline.chomp
}
file.close()
```

In that example, you will read an existing file name text.txt in drive D. The program will read the file each lines, save it in variable thisline and print variable thisline on the screen after chomped, and finally close the file.

6.2 Getting value from file

In previous section, you already learned how to print all lines in the file to the screen. Now, you will learn how to get the value from the file. For example, if in the file contains "ABC#12", and you want to print or get the value of "ABC" and "12", you can code:

```
file = File.new("D:\\text.txt", "r")
file.each {|thisline|
  print thisline[0..thisline.index("#")-1]
  print " "
  puts thisline[thisline.index("#")+1..thisline.length]
}
file.close()
```

As you can see, you can use range index to get the specified value from variable thisline. In the example above, "print thisline[0..thisline.index("#")-1]" will print thisline from index 0 to index where character "#" located and subtract it by one. Then, "thisline[thisline.index("#")+1.. thisline.length]" will print thisline from index where character "#" located and added it by one to the end of this line.

For more information, see http://www.tutorialspoint.com/ruby/ruby_input_output.htm.

6.3 The Case

In this case, you will need main looping, which ended when user don't want to exit (input 4). You must validate them using selection. Next, in the main loop, you will need selection to the inputted number. There are four possible cases.

When input is 1, read the file and print the product list to the screen (using mode "r"). We will split each several times here. First, we separate the string from the thisline before ";" sign, save it as variable productname and remainingtext. Second, we separate the string from the remainingtext before ";" sign, save it as variable productprice and productstock.

When input is 2, append the file with the new product that inputted by user (using mode "a").

When input is 3, read the file (using mode "r", same as when input 1) and save the product list to a new variable if it is not the product that the user want to delete. Next, the program will write that new variable (using mode "r").

When input is 4, it just shows a message.

For more detail, you can learn the code below by yourself.

```
1     begin
2       puts 'Hello, Manager!'
3       puts 'What do you want to do?'
4       puts '1. View product list'
5       puts '2. Add a new product'
6       puts '3. Delete an existing product'
7       puts '4. Exit'
8       print '> '
9       input = gets.to_i
10
11      case input
12      when 1
13        line = 1
14        puts "PRODUCT LIST"
15        puts "------------"
16        puts "No. ".ljust(6) << "Product name".ljust(20) << '    '<<
      "Price".rjust(6) << "Stock".rjust(6)
17
18        file = File.new('D:\\text.txt', 'r')
19        file.each {|thisline|
20          productname = thisline[0..thisline.index(';')-1]
21          remainingtext  = thisline[thisline.index(';')+1..thisline.length]
```

```
22
23          productprice = remainingtext[0..remainingtext.index(';')-1]
24          productstock = remainingtext[remainingtext.index(';')+1..
    remainingtext.length]
25
26          puts (line.to_s.<<(".").ljust(6) << productname.ljust(20) << 'Rp.'<<
    productprice.rjust(6) << productstock.rjust(6)
27          line += 1
28        }
29        file.close()
30        print 'Press Enter to continue.'
31      when 2
32        begin
33          puts 'Name of the product:'
34          print '> '
35          productname = gets.chomp
36        end until productname.length > 0
37
38        begin
39          puts 'The price of ' << productname << ' in rupiahs:'
40          print '> '
41          productprice = gets.chomp
42          begin
43            isnumber = true
44            Integer(productprice)
45          rescue
46            isnumber = false
47          end
48        end until isnumber == true and productprice.to_i > 0
49
50        begin
51          puts 'The amount of ' << productname << ' you own:'
52          print '> '
53          productstock = gets.chomp
54          begin
55            isnumber = true
56            Integer(productstock)
57          rescue
58            isnumber = false
59          end
60        end until isnumber == true and productstock.to_i > 0
61
62        file = File.new('D:\\text.txt', 'a')
63        file.puts productname << ";" << productprice << ";" << productstock
64        file.close()
65
66        puts 'Product added successfully.'
67      when 3
68        puts 'Type the product name that you want to delete:'
69        print '> '
70        deletedproductname = gets.chomp
71        deletedproducts = 0
72        newfilestring = ''
73
74        file = File.new('D:\\text.txt', 'r')
```

```
75        file.each {|thisline|
76          productname = thisline[0..thisline.index(';')-1]
77          remainingtext  = thisline[thisline.index(';')+1..thisline.length]
78
79          productprice = remainingtext[0..remainingtext.index(';')-1]
80          productstock = remainingtext[remainingtext.index(';')+1..
    remainingtext.length]
81
82          if productname != deletedproductname
83            newfilestring += productname << ";" << productprice << ";" <<
    productstock
84          else
85            deletedproducts += 1
86          end
87        }
88        file.close()
89
90        if deletedproducts > 0
91          file = File.new('D:\\text.txt', 'w')
92          file.print newfilestring
93          file.close()
94          puts 'Product deleted successfully.'
95        else
96          puts 'No such product name exists.'
97        end
98      when 4
99        print 'Thank you. See you later.'
100     else
101       print 'Invalid input.'
102     end
103     gets
104     puts
105   end until input==4
```

This is the end of chapter six.

This is the end of our training; I hope it can help you to understand the basic of Ruby.

CHAPTER 1:
INTRODUCTION

1.1 What is Ruby?

Ruby is a dynamic, open source interpreted programming language which you don't need to compile the source code first if you want to run it. Ruby is focus on simplicity and productivity. It has an elegant syntax that is natural to read and easy to write.

It is used on many programs, including NASA Langley Research Center simulator, Motorola simulator, Google Sketch-Up, RPG Maker, and many more. For more information, see http://www.ruby-lang.org/en/documentation/success-stories/.

1.2 Compiled versus Interpreted Language

From this training module subtitle "Compile No More!" you might already notice that in Ruby Programming Language, you don't need to compile the source code. This kind of programming language is called as interpreted language. So what's the different?

Compiled Language

A compiled language is a programming language whose implementations are typically compilers which generate machine code from source code.

Some of compiled languages are: C (including C++ and Objective C), FORTRAN, Pascal, and Basic.

Interpreted Language

An interpreted language is a programming language whose implementations are typically interpreters which executing the source code step-by-step where no translation takes place.

Some of interpreted languages are: Ruby, Python, and Scripts (including Command Line Script, Java script, and J-script).

1.3 Why using Interpreted Language?

Interpreting a language gives implementations some additional flexibility over compiled implementations. Features are often easier to implement in interpreters than in compilers include

- platform independence
- reflection and reflective use of the evaluator (e.g. a first-order *eval* function)
- dynamic typing (variant type)
- smaller executable program size (since implementations have flexibility to choose the instruction code)
- dynamic scoping

1.4 History

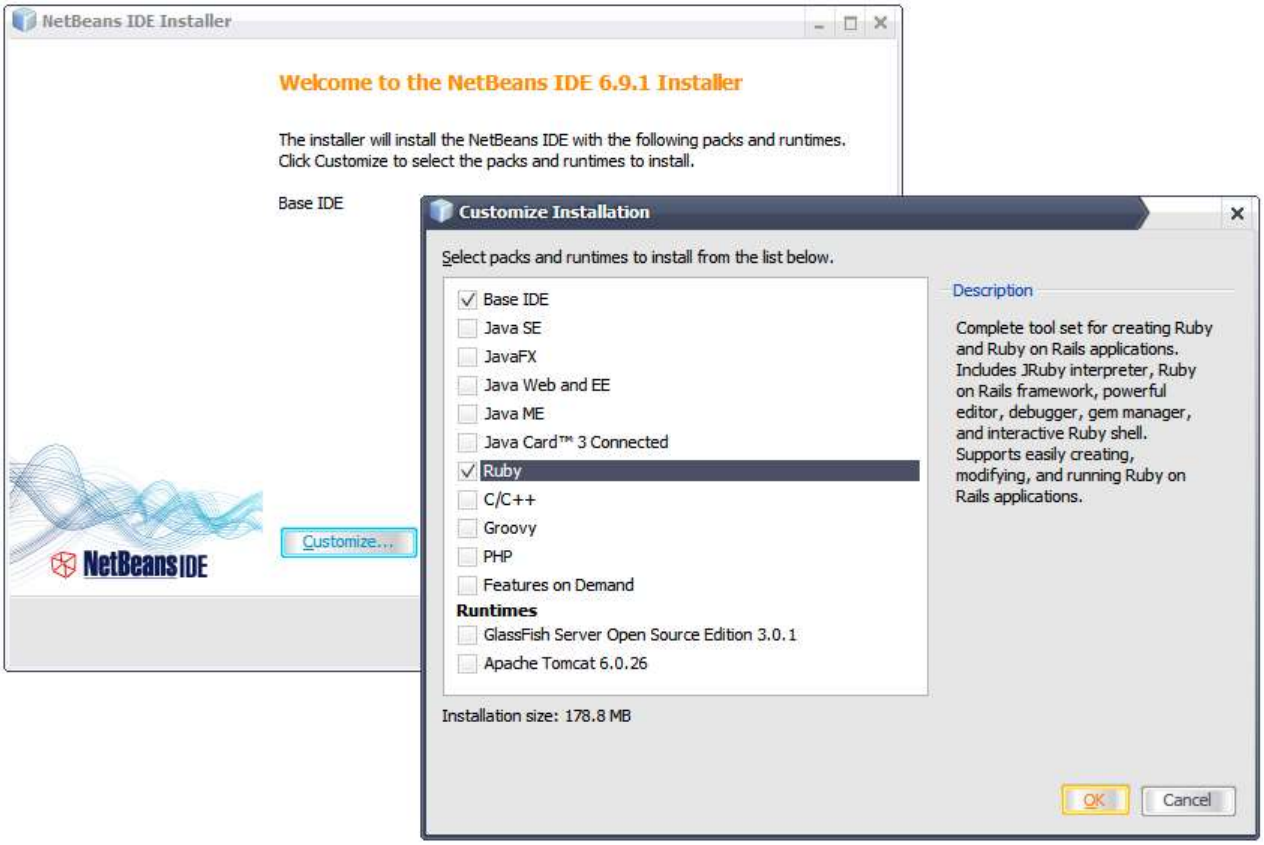
Ruby was conceived on February 24, 1993 by Yukihiro Matsumoto who wished to create a new language that balanced functional programming with imperative programming. Matsumoto has stated, "I wanted a scripting language that was more powerful than Perl, and more object-oriented than Python. That's why I decided to design my own language".

At a Google Tech Talk in 2008 Matsumoto further stated, "I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language."

1.5 System requirement

In order to make the computer can run a Ruby Programming Language, you must install an IDE. There are many IDE which supports Ruby now, like NetBeans, RubyForge, RadRails, RubyMine, and ActiveState Komodo. In this module, we will use NetBeans with JRuby interpreter which is more popular and often used.

When installing NetBeans, don't forget to tick the Ruby component, or just go with the full installation.

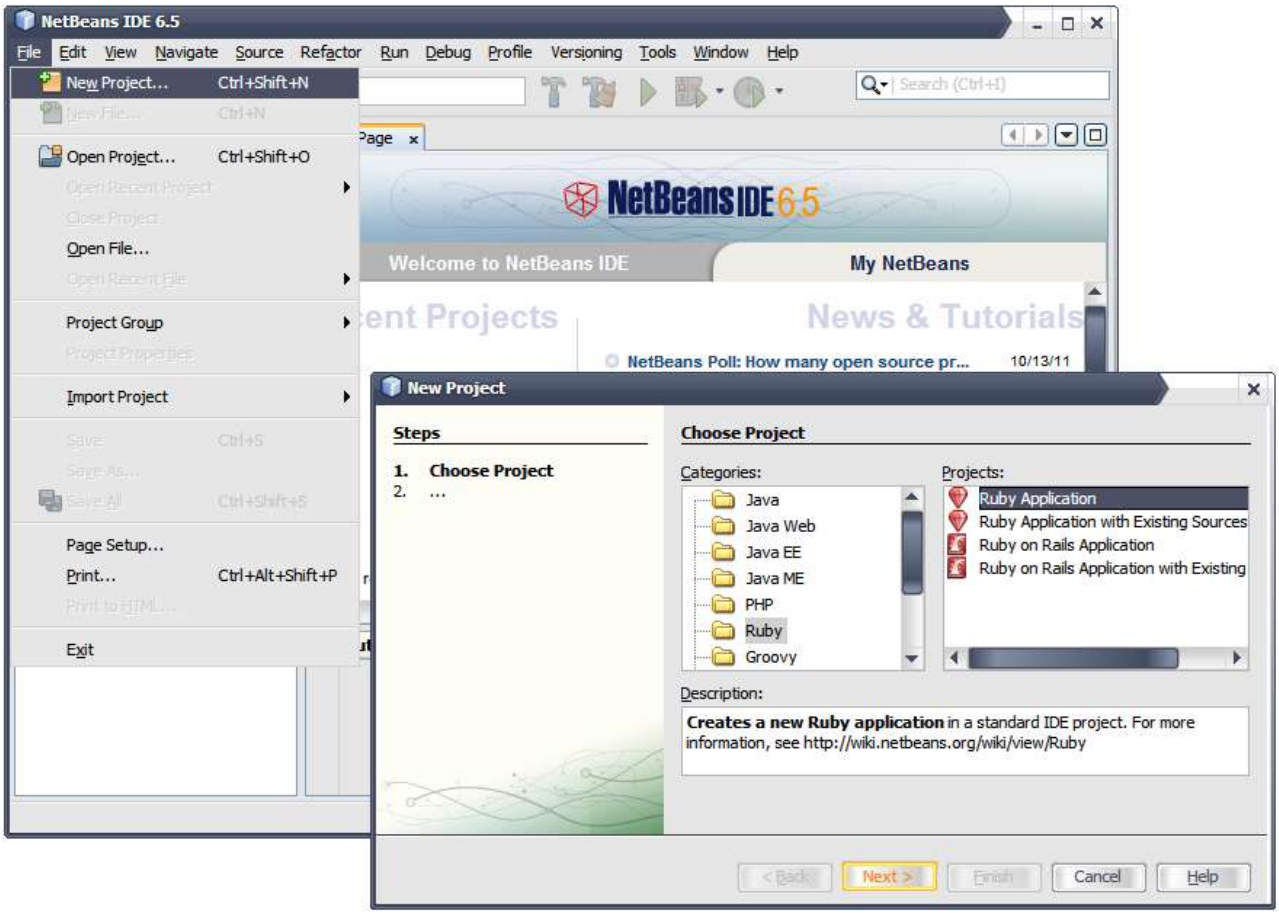


Note: NetBeans release 7.0 and above no longer supports Ruby (and Rails). You can check if latest separate NetBeans IDE Bundle for Ruby is available or not at http://wiki.netbeans.org/RubySupport.

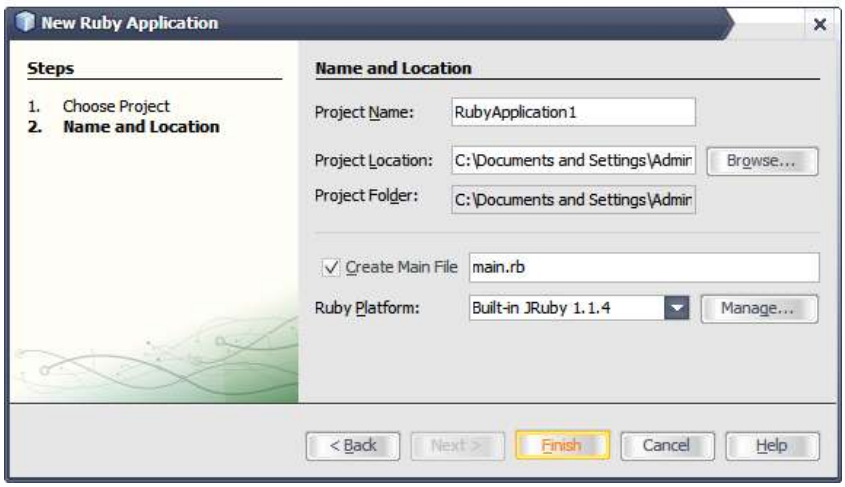
CHAPTER 2:
GETTING STARTED

2.1 Creating a new project

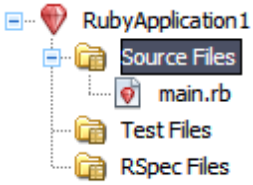
To get started, open NetBeans. Click New Project in File menu, and select Ruby Application in New Project Window.



In the next window, specify your project name, location, main file, and platform. Hit Finish then your project will be created.



NetBeans will automatically create some folders and files. You can leave them or delete some of them. To run a Ruby project, at least you must have following folders and files in your project.



2.2 How to comment

In Ruby programming language you can use =begin to begin commenting some lines of syntax, and =end to end it. You can also use # in front of the text which you want to comment until the end that line.

Example:

```
=begin
 Comment line 1
 Comment line 2
=end

#Single line comment
```

CHAPTER 3:
CREATE AN ANGRY BOSS PROGRAM

In this part of the training, you will learn how to make a program (as the boss) that ask the user (as the employee) what is his/her name, ask what he/she want, and the fired him/her.

Look at this example:

```
Okay what's your name?
> Matz
What do you want now, Matz?
> I want a raise.
WHADDAYA MEAN "I WANT A RAISE."?!?  YOU'RE FIRED!!
```

5.5 Output

Now, before we begin, let's learn about output in Ruby.

To print a text into the console screen, you can use these command.

- puts → print text and end it with line break.
- print → print text without a line break.
- printf → print text with specific format. To learn more about this format, see http://sharkysoft.com/archive/printf/docs/javadocs/lava/clib/stdio/doc-files/specification.htm.

You can use << to concatenated the text. For puts and print, you can these methods to modify how the program shows the text:

- ljust width → create width times spaces, and print it in the left
- rjust width → create width times spaces, and print it in the right
- center width → create width times spaces, and print it in the center

For example, if you code “puts "Ruby".center 80”, it will print “Ruby” in the center of the console screen. The program will print 80 spaces and overwrite the “Ruby” text in the center. Note: you can use single quote or double quote to set the text.

You can also change the case with these methods:

- upcase → convert the text to uppercase
- downcase → convert the text to lowercase
- swapcase → invert the case, the caps will become lowercase, and the opposite.
- capitalize → capitalize the first letter and lowercase the others

5.6 Variable

To make a variable just write any plain, lowercase word. Variables may consist of letters, digits and underscores. With variables, you give a nickname to something you use frequently, or you can set it with value from user input.

Example: `pencil_price = 1000`

5.7 Input

After you learned output, let’s go on to how to make a user input in Ruby.

To scan an input, you can use gets command which get the input in string. But it will also retrieve the line break. So, maybe you will need the Chomp method which will remove the line break.

Example: `inputted_name = gets.chomp`

5.8 The Case

So, with that information you can make the program mentioned in the first part. If you are still don’t know it, here’s the code:

```
1 puts "Okay what's your name?"
2 print "> "
3 name = gets.chomp
4
```

```
5 puts "What do you want now, " << name << "?"
6 print "> "
7 want = gets.chomp
8
9 puts 'WHADDAYA MEAN "' << want.upcase << '"?!?  YOU' << "'RE FIRED!!"
```

The first line of code will print “Okay what’s your name?” in the screen with line break.

The second will print a greater than sign and a space, without line break.

The third will take value from what the user inputted, chomped it, and set it to variable name.

The fifth will print “What do you want now, ” followed by variable name, a question mark, and a line break.

The sixth will print a greater than sign and a space, without line break.

The seventh will take value from what the user inputted, chomped it, and set it to variable want.

The last will print “WHADDAYA MEAN "” followed by variable want in uppercase, and then print “?!? YOU'RE FIRED!!”.

This is the end of chapter three.

CHAPTER 4:
CREATE AN INTERACTIVE MENU

In this part of the training, you will learn how to make a program with main menu. And the user will have to choose want he/she want. After selecting menu one and input his/her name, it will back to the main menu. But if the user select menu two, it will print a greeting and end the program.

Look at this example:

```
1. Input a name
2. Exit
> 1
What's your name?
> Matz
1. Input a name
2. Exit
> 1
What's your name?
> John Doe
1. Input a name
2. Exit
> 2
Thank you for using this program
```

4.4 Looping

Now, before we begin, let's learn about looping in Ruby.

In Ruby, there are many ways to make a loop. There are times, for, each, while, and until. Times is the simplest one to do loop. For example, if you code "2.times {puts "Hello"}", Ruby will understand it, and puts word Hello on the screen two times.

In this case, we will use until. The format is type “begin”, and followed by the statement(s) you want to loop, after that type “end until” followed by the condition.

For more information see http://www.tutorialspoint.com/ruby/ruby_loops.htm, or the answer code in section 4.3.

4.5 Selection

Now, let’s learn about selection in Ruby.

In Ruby, there are many ways to make a loop. There are if, case, and ternary operator. If is the simplest one to make a selection. For example, if you code “puts "Your number is odd." if number%2==1”, Ruby will understand it, and only puts word “Your number is odd.” on the screen if the variable number modulus by two is one, which means an odd number.

In this case, we will use case. The format is type “case”, and followed by the condition and statement(s) you want to do, after that type “end”. Use “when” to specify the condition, then you can follow it with the statement(s) in the next line(s).

For more information see http://www.tutorialspoint.com/ruby/ruby_if_else.htm, or the answer code in section 4.3.

4.6 The Case

So, here’s the code:

```
1   begin
2     puts "1. Input a name"
3     puts "2. Exit"
4     print "> "
5     input = gets.to_i
6
7     case input
8     when 1
9       puts "What's your name?"
10      print "> "
```

```
11      gets
12    when 2
13      puts "Thank you for using this program."
14    else
15      puts "Invalid input."
16    end
17  end until input == 2
```

The first and the last line of codes will mark the loop. The program will do the loop until the value of variable input is 2. Inside the loop it will print “1. Input a name”, “2. Exit”, and “> ” (line 2-4). Later, in line 5, it will scan what the user inputted, converted it to integer (using to_i), and save it to variable input.

You can use to_i to convert the scanned input into integer, to_s to convert the scanned input into string, or to_f to convert the scanned input into float.

The next part it (line 7-16) will do a selection to the variable input. When variable input value is 1, it will print “What's your name?”, and “> ” (line 9-10). Later, in line 11, it will scan what the user inputted. But when variable input value is 2, it will print “Thank you for using this program.” Else, it will print “Invalid input.”.

This is the end of chapter four.

CHAPTER 5:
CREATE A FILE WRITER

In this part of the training, you will learn how to make a program which can write a file.

Look at this example:

```
Type your document file and location (e.g. "D:\text.txt") below!
> D:\text.txt
Type your document now. Type "::end::" to finish typing:
> Hello
> World!
> I'm creating this file using my own program.
> ::end::
File saved.

Type a new document again? [Yes/No]
> AAA
Invalid input.
Type a new document again? [Yes/No]
> No
```

5.3 Writing file

Now, let's learn about file writing in Ruby.

To write a file in Ruby, you can use new method in File class (File.new), and specify the access right mode, "w" in this case. "w" mode will write a new file or overwrite existing file. If you want to append something to the file, you can use "a" mode. To write to the file, you still can use puts, print, and printf. After that, don't forget to close the file.

For example, you can code:

```
file = File.new("D:\\text.txt", "w")
file.puts("Hello World!")
file.close()
```

In that example, you will create a new file name text.txt in drive D and you can write it. The program will write "Hello World!" to the file, and finally close the file.

For more information, see http://www.tutorialspoint.com/ruby/ruby_input_output.htm.

5.4 The Case

In this case, you will need main looping, which ended when user don't want to type a new document again. You must validate them using selection. Next, in the main loop, you will need another loop that will continue append the file until user type "::end::". Now, try to make the program by yourself!

This is the end of chapter five.

CHAPTER 6:
CREATE A PRODUCT LIST MANAGER

In this last part of the training, you will learn how to make a product list manager program which can read and write a file, and also use all of the syntax you learn before.

Look at this example:

```
Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 1
PRODUCT LIST
------------------
No      Product name        Price         Stock
1.      Book                Rp. 3000      10
2.      Pencil              Rp. 1000       9
3.      Eraser              Rp.  500       5
Press Enter to continue.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 2
Name of the product:
> Pen
```

```
The price of Pen in rupiahs:
> 2500
The amount of Pen you own:
> 20
Product added successfully.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 3
Type the product name that you want to delete:
> Book
Product deleted successfully.

Hello, Manager!
What do you want to do?
1. View product list
2. Add a new product
3. Delete an existing product
4. Exit
> 4
Thank you. See you later.
```

6.4 Reading file

Now, let's learn about file reading in Ruby.

Same as writing file, to read a file in Ruby, you can use new method in File class (File.new), and specify the access right mode, "r" in this case. But, when you want to get the value of the line you can't use 'gets', you need using 'each' instead.

For example, you can code:

```
file = File.new("D:\\text.txt", "r")
file.each {|thisline|
  puts thisline.chomp
}
file.close()
```

In that example, you will read an existing file name text.txt in drive D. The program will read the file each lines, save it in variable thisline and print variable thisline on the screen after chomped, and finally close the file.

6.5 Getting value from file

In previous section, you already learned how to print all lines in the file to the screen. Now, you will learn how to get the value from the file. For example, if in the file contains "ABC#12", and you want to print or get the value of "ABC" and "12", you can code:

```
file = File.new("D:\\text.txt", "r")
file.each {|thisline|
  print thisline[0..thisline.index("#")-1]
  print " "
  puts thisline[thisline.index("#")+1..thisline.length]
}
file.close()
```

As you can see, you can use range index to get the specified value from variable thisline. In the example above, "print thisline[0..thisline.index("#")-1]" will print thisline from index 0 to index where character "#" located and subtract it by one. Then, "thisline[thisline.index("#")+1.. thisline.length]" will print thisline from index where character "#" located and added it by one to the end of this line.

For more information, see http://www.tutorialspoint.com/ruby/ruby_input_output.htm.

6.6 The Case

In this case, you will need main looping, which ended when user don't want to exit (input 4). You must validate them using selection. Next, in the main loop, you will need selection to the inputted number. There are four possible cases.

When input is 1, read the file and print the product list to the screen (using mode "r"). We will split each several times here. First, we separate the string from the thisline before ";" sign, save it as variable productname and remainingtext. Second, we separate the string from the remainingtext before ";" sign, save it as variable productprice and productstock.

When input is 2, append the file with the new product that inputted by user (using mode "a").

When input is 3, read the file (using mode "r", same as when input 1) and save the product list to a new variable if it is not the product that the user want to delete. Next, the program will write that new variable (using mode "r").

When input is 4, it just shows a message.

For more detail, you can learn the code below by yourself.

```
1     begin
2       puts 'Hello, Manager!'
3       puts 'What do you want to do?'
4       puts '1. View product list'
5       puts '2. Add a new product'
6       puts '3. Delete an existing product'
7       puts '4. Exit'
8       print '> '
9       input = gets.to_i
10
11      case input
12      when 1
13        line = 1
14        puts "PRODUCT LIST"
15        puts "------------"
16        puts "No. ".ljust(6) << "Product name".ljust(20) << '    '<<
      "Price".rjust(6) << "Stock".rjust(6)
17
18        file = File.new('D:\\text.txt', 'r')
19        file.each {|thisline|
20          productname = thisline[0..thisline.index(';')-1]
21          remainingtext  = thisline[thisline.index(';')+1..thisline.length]
```

```
22
23          productprice = remainingtext[0..remainingtext.index(';')-1]
24          productstock = remainingtext[remainingtext.index(';')+1..
    remainingtext.length]
25
26          puts (line.to_s.<<(".").ljust(6) << productname.ljust(20) << 'Rp.'<<
    productprice.rjust(6) << productstock.rjust(6)
27          line += 1
28        }
29        file.close()
30        print 'Press Enter to continue.'
31      when 2
32        begin
33          puts 'Name of the product:'
34          print '> '
35          productname = gets.chomp
36        end until productname.length > 0
37
38        begin
39          puts 'The price of ' << productname << ' in rupiahs:'
40          print '> '
41          productprice = gets.chomp
42          begin
43            isnumber = true
44            Integer(productprice)
45          rescue
46            isnumber = false
47          end
48        end until isnumber == true and productprice.to_i > 0
49
50        begin
51          puts 'The amount of ' << productname << ' you own:'
52          print '> '
53          productstock = gets.chomp
54          begin
55            isnumber = true
56            Integer(productstock)
57          rescue
58            isnumber = false
59          end
60        end until isnumber == true and productstock.to_i > 0
61
62        file = File.new('D:\\text.txt', 'a')
63        file.puts productname << ";" << productprice << ";" << productstock
64        file.close()
65
66        puts 'Product added successfully.'
67      when 3
68        puts 'Type the product name that you want to delete:'
69        print '> '
70        deletedproductname = gets.chomp
71        deletedproducts = 0
72        newfilestring = ''
73
74        file = File.new('D:\\text.txt', 'r')
```

```
75        file.each {|thisline|
76          productname = thisline[0..thisline.index(';')-1]
77          remainingtext  = thisline[thisline.index(';')+1..thisline.length]
78
79          productprice = remainingtext[0..remainingtext.index(';')-1]
80          productstock = remainingtext[remainingtext.index(';')+1..
    remainingtext.length]
81
82          if productname != deletedproductname
83            newfilestring += productname << ";" << productprice << ";" <<
    productstock
84          else
85            deletedproducts += 1
86          end
87        }
88        file.close()
89
90        if deletedproducts > 0
91          file = File.new('D:\\text.txt', 'w')
92          file.print newfilestring
93          file.close()
94          puts 'Product deleted successfully.'
95        else
96          puts 'No such product name exists.'
97        end
98      when 4
99        print 'Thank you. See you later.'
100     else
101       print 'Invalid input.'
102     end
103     gets
104     puts
105   end until input==4
```

This is the end of chapter six.

This is the end of our training; I hope it can help you to understand the basic of Ruby.

BIBLIOGRAPHY

Web

http://www.ruby-lang.org/

http://en.wikipedia.org/wiki/Ruby_(programming_language)

http://rubylearning.com/satishtalim/tutorial.html

http://www.tutorialspoint.com/ruby/

http://www.troubleshooters.com/codecorn/ruby/basictutorial.htm

Book

Why the lucky stiff. 2005. *Why's (poignant) Guide to Ruby*. A-Press.