COM1001 Introduction to Software Engineering

Ruby Gems

Ruby concepts introduced in this lab: *Modules* · *Namespaces* · *Blocks* · *Iterations* · *Gems* · *Bundler*

1 Modules

Modules are a way of grouping together methods, classes, and constants. This is great for re-use, and a main advantage is that it prevents name clashes. That is, it is possible to define methods with the same name in different modules, as each module provides its own *namespace*.

```
module Example
  MY_CONSTANT = 7

  def Example.some_method(x)
    return MY_CONSTANT * x
  end
end
```

Note that module names need to start with a capital letter To use either the method `some_method` or the constant `MY_CONSTANT` outside of the `Example` module, we need to include the module name, as this is the namespace of the method and the constant:

```
x = Example.some_method 2
y = Example::MY_CONSTANT
```

In order to access these from another program, we need to use the `require` keyword, which takes the filename (not the module name) as a parameter. It is not necessary to include the ".rb" part:

```
require "example"
x = Example.some_method 3
```

Note that if your module is in the same directory as the code from which you are trying to access it, you may need to tell the Ruby interpreter that it should look for modules in the current directory:

```
$LOAD_PATH << '.'
require "example"
x = Example.some_method 3
```

Modules can be nested, in which case the operator "`::`" is used to separate namespaces, and the method needs to be explicitly assigned to its module:

```
module Foo
  module Bar
    def Bar.foo_bar
      # ...
    end
  end
end

Foo::Bar.foo_bar
```

(If you are already familiar with object-oriented programming, this may seem similar to the concept of classes. However, modules provide only behaviour but no state.)

2 Ruby Gems

RubyGems is a system for managing Ruby software libraries. A Ruby module packaged in this manner is called a gem. When you find Ruby software you want to use in a project, gems offer a means of downloading, installing and managing the software. To find out about available gems, you can use the official RubyGem website, which provides search functionality and links to each gem's project website, documentation, etc:

`https://rubygems.org/`

2.1 Ruby Gems on the Command Line

Remember that you can launch the command line prompt on Windows by selecting 'Start Command Prompt with Ruby' from the 'Ruby 2.1.6' entry in the start menu. To install gems on your machine on the command line, you need to use the `gem` tool. The `gem` tool takes a command as first parameter. For example, the "`search`" command allows searching for gems. To search for all gems related to Twitter, you could try:

```
gem search twitter
```

To install a gem on the command line, type the following:

```
gem install name of gem
```

To check which gems you have already installed, use

```
gem list
```

To find out about further commands provided by the `gem` tool, consult the help:

```
gem help
```

To get help on how to use an installed gem, you can find documentation to all gems online at `http://www.rubydoc.info/gems`.

2.2 Managing Gem Dependencies with Bundler

Although installing gems isn't difficult, it can become tedious for all members of a development team to manually install gems whenever a new dependency is required. Bundler[1] is a Ruby tool to manage the dependencies of a project in a convenient way. Bundler itself is a gem, so it can be installed using:

```
gem install bundler
```

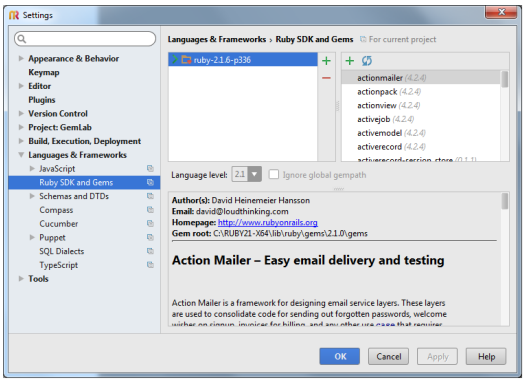
Once Bundler is installed, you can use it by creating a text file in the root directory of your project and naming it `Gemfile`. This file contains a list of all required gems, and optionally also their versions. Given a `Gemfile`, one can install all required gems easily using the following command:

```
bundle install
```

This command installs the required gems, and creates a file `Gemfile.lock` in which details of the last installation are captured. Both files, `Gemfile` and `Gemfile.lock`, should be considered part of your project now. We will have a look at an example `Gemfile` later in this exercise.

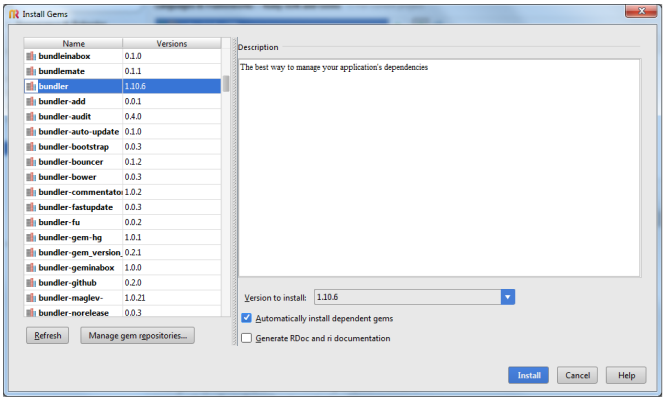
2.3 Ruby Gems in RubyMine

RubyMine also provides a gem manager. You can find it by opening the Project Settings dialog in RubyMine (File → Settings → Languages & Frameworks → Ruby SDK and Gems).



[1] http://bundler.io/

In this dialog you can browse the installed gems as well as the available gems. To install a gem, click the "+" button (the one on the righthand side next to the list of installed gems) and select the gem from the list.



3 A Google Search Text Interface

To practice the use of Gems and Bundler, we will be writing a small Ruby script that performs search requests on Google web search, and shows the results in a nice text table. Incidentally, this also requires the use of iterators, as discussed in the last lecture.

3.1 Gem Dependencies

We will require three gems:

- `google-search` – an API to send queries to Google Search.
- `htmlentities` – decoding and encoding of HTML entities.
- `text-table` – pretty printing of text tables.

1. Make sure you have the bundler gem installed. You can either do this on the command line:

   ```
   gem install bundler
   ```

   Or alternatively, you can use RubyMine's gem manager.

2. Create a new RubyMine project.

3. In RubyMine, select "Tools" → "Bundler" → Init. Note that the "Bundler" menu item will only appear if the bundler gem is installed. This menu option will create a template Gemfile.

4. Edit the Gemfile, and add the three dependency gems as follows:

   ```
   source 'https://rubygems.org'
   gem 'google-search'
   gem 'htmlentities'
   gem 'text-table'
   ```

5. Now run install on the Bundler to install the three required gems. Select "Tools" → "Bundler" → Install. (Make sure that the output confirms this succeeded.)

3.2 Implementation

1. This program will make use of all three gems just installed. To use them in our program, we need to tell the Ruby interpreter that we plan to use them. This is achieved with the `require` command. Typically, the list of required gems is provided at the beginning of a Ruby file, so at the top of the script insert the following commands:

```
require "google-search"
require "htmlentities"
require "text-table"
```

2. Add a method `do_search` that takes two parameters: A string (the search query) and a number (the number of results to return). On this query string, the method should perform a Google web search, and print up to the number of search results specified by the second parameter in a nice table.

   The google-search gem can be used by creating a new `Google::Search::Web` object, and setting its `query` field to the search terms:

```
search = Google::Search::Web.new
search.query = "Some query string"
```

   The `search` object can now be queried for search results using the `each` method. This is a standard way to iterate in Ruby, and is actually preferred to traditional `for` loops. The `each` method takes a *block* as parameter. A block in Ruby is piece of code that can be passed as method parameters, and the `each` method will invoke this piece of code on every item of the iteration. Blocks can enclosed in curly braces or `do-end` constructs; the convention is to use braces for single-line blocks and `do-end` for multi-line blocks. For example, consider the following invocation of `each` on an array object:

```
[1, 2, 3].each { puts "Hello" }
```

   The array contains three elements (1, 2, and 3). The block consists only of the `puts` statement. The `each` method now invokes this block once for each of its three elements, resulting in the following output:

```
Hello
Hello
Hello
```

   Blocks can take parameters, which are specified at the beginning of the block between pipe symbols:

```
[1, 2, 3].each { |x| puts "Current value: #{x}" }
```

   This block defines one parameter `x`, and the `each` method of the array passes the current value of the iteration to the block, such that the following output is produced:

```
Current value: 1
Current value: 2
Current value: 3
```

   We can iterate over Google search results in a similar way, by passing a block to the `each` method of the `search` object. For example, the following example calls `each` with a block that defines one parameter `item` which is printed to the standard output. The `each` method then invokes this block for every search result.

```
search.each do |item|
  puts item.title
end
```

3. For each element of the search result, print the title and the URI, which you can access using `item.title` and `item.uri`, respectively, if your result variable is called `item`.

4. Make sure you try out your program every now and then, instead of implementing the whole lab sheet before running the program the first time. The longer you wait for testing your program, the more difficult it will be to fix any problems that may occur. For example, at this point you could try calling the `do_search` method with an example query and see if it works so far.

5. To create a nicer looking output, let's now use the `text-table` gem to format the output. To use `text-table`, you first need to create a new `Text::Table` object. On this, you can then modify the `rows` field, which contains a two-dimensional array. For example, consider the following Ruby code:

```
table = Text::Table.new
table.head = ['Title', 'URL']
table.rows << ['Some title', 'Some URL']
table.rows << ['Other title', 'Other URL']

puts table.to_s
```

   The `head` is a one-dimensional array with the length of the number of columns of the table, containing the column names. The `rows` field of the table is a two-dimensional array; that is, it is an array where each element is itself an array. The << operator appends an item to an array; in this case the item is another array (`['Some title', 'Some URL']`). You can repeatedly use this operator, and each time the array will grow by one element. (Alternatively, you could also use the `push` method, which does the same just with normal method syntax.) Finally, `to_s` converts the table to a string. The output of this script will be:

```
+--------------+------------+
|    Title     |    URL     |
+--------------+------------+
| Some title   | Some URL   |
| Other title  | Other URL  |
+--------------+------------+
```

6. When printing `item.title` you will notice that there can be HTML entities[2] included in the text. As we want to print to the console rather than a web browser, we need to decode these characters in the title string before printing it. For this, we can use the `htmlentities` gem. For example, the following code creates a decoder, and creates a new string `output` that contains the text from `input`, but with all HTML entities decoded to regular characters:

```
decoder = HTMLEntities.new
output = decoder.decode(input)
```

   For example, a Google search for "Ruby Programming" includes the following title with HTML entities:

```
Programming Ruby 1.9 &amp;amp; 2.0: The Pragmatic Programmers&amp;#39; Guide
```

   After applying the decoder, the string looks like this:

```
Programming Ruby 1.9 &amp; 2.0: The Pragmatic Programmers' Guide
```

7. Now add some code to ask the user for a query string and the number of items to return, invoke the `do_search` method with this query string and number, and call `print_results` with the result.

If this is too easy for you, refactor the code to have a method `do_search` that returns an array, and a method `print_results` that does the output. Try and change the layout and border symbols of the table by reading the text-tables documentation. Ensure that the column width is exactly 35 characters.

[2] http://www.w3schools.com/html/html_entities.asp