CIT 483/583 Lab – Object Storage

**Instructions**

Open and save this file in any MS word-compatible format as Lab08\_*Firstname*\_*Lastname*.<ext> and place your answers in that document. Do a **Save-As** and retain all of my content. Keep the document safe in case your submission fails, or you discover an error prior to the due date and wish to re-submit. Submit your document to the Lab08 dropbox in Canvas. The due date and any other pertinent information are noted in the Canvas item.

This lab should be completed on students.cs.nku.edu as a reference implementation and as a means of testing your answers.

**Place the answers in or immediately following each question and make sure your answers stand out from the questions by using a different font color.**

1. In order to make our program02.rb code more portable, we can refactor it. First, copy program02\_solution.rb from my home directory to suitable work place under your home directory. A suggested name for a subdirectory would be lab08, but it's your choice. List the contents of that directory.

$ ls # in the directory you copied the file

1. Now, split it into 4 files:

* generate\_password.rb - contains only the module definition for GeneratePassword
* user\_account.rb - contains the class definition for UserAccount
* person.rb - contains the class definition for Person
* program02\_alt.rb - contains the rest of the original program02\_solution.rb code

List the new contents of that directory.

$ ls # in the directory you copied the file

1. Edit program02\_alt.rb to add three load directives near the top of the file:

load 'generate\_password.rb'

load 'user\_account.rb'

load 'person.rb'

You should be able to run program02\_alt.rb at this point, as long as all of the files are named correctly and are in that same directory. Capture only the last 10-12 lines to show that you got it to run.

$ ruby program02\_alt.rb

1. Now we can work with these objects in IRB without having to paste large blocks of text into it, and

we can see how *load* and *require* work in a bit more detail. Capture all output.

$ irb --simple-prompt

>> load 'generate\_password.rb'

>> require 'generate\_password.rb' # ERROR

>> require './generate\_password.rb' # needs explicit path

>> $: # check LOAD\_PATH

>> $LOAD\_PATH # alias for $:

>> $: << '.' # add the current directory to $LOAD\_PATH

>> require 'user\_account.rb' # now Ruby knows to look at .

>> load 'person.rb'

>> load 'person.rb' # always true, even if already loaded

>> require 'user\_account.rb' # false after first time

**If you restart IRB, you will need to load all three files to continue.**

1. Now we can create a few UserAccount and Person objects for testing. If there is no prompt shown (>>), just paste blocks directly into IRB and capture the commands and output as one block.

p1 = Person.new(799)

p1.first\_name = 'Tim'

p1.middle\_initial = ' '

p1.last\_name = 'McCord'

p1.suffix = ' '

p1.department\_id = 909

p1.role = 'E'

p1.phone\_number = '111-222-3333'

p1.email\_address = 'mccordt@nku.edu'

ua1 = UserAccount.new

ua1.user\_id = "mccordt"

ua1.gecos = "Tim McCord"

ua1.groups = ["root", "other"]

ua1.home\_dir = "/home/mccordt"

# With the workaround in UserAccount this can be done

ua1.password = ua1.generate\_plain\_text

ua1.account\_owners = [p1, Person.new(845)]

ua1.access\_granted\_date = Time.now

ua1.access\_suspended\_date = Time.now + 365

>> ua1 # check the object

>> p1 # check the object

1. Create at least one more UserAccount and Person object, using your own name if you like. Call them ua2 and p2. Show your commands and output as two blocks.

>> ua2 # check the object

>> p2 # check the object

**If you restart IRB after this step, you will need to load all three .rb files, and paste the blocks the create ua1, ua2, p1, and p2 to recreate those objects.**

1. Now we can serialize and save our objects by creating a CSV file. You may or may not have seen this before, but it's fairly intuitive. Add the data for ua2 to the block below, and then paste the whole block. If you make a mistake, the file gets recreated every time, so you can't go too far wrong.

>> require 'csv'

CSV.open('user\_accounts.csv', 'w') do |writer|

# write the header row

writer << %w[user\_id gecos home\_dir groups account\_owners password access\_granted\_date access\_suspended\_date]

# write the data row(s)

writer << [ua1.user\_id, ua1.gecos, ua1.home\_dir, ua1.groups, ua1.account\_owners, ua1.password, ua1.access\_granted\_date, ua1.access\_suspended\_date]

**# add ua2 data here using the same syntax**

end

>> `cat user\_accounts.csv` # show the results

1. Now we can attempt to read the data back and recreate a UserAccount object using the

rows = []

CSV.foreach('user\_accounts.csv', :headers=>true) do |reader|

rows << reader[0..-1]

end

>> rows.each { |row| puts row } # check content

>> user\_accounts = []

rows.each do |row|

temp = UserAccount.new

temp.user\_id = row[0]

temp.gecos = row[1]

**# do the same for the remaining attributes of temp**

user\_accounts << temp # add temp to the array

end

>> user\_accounts.each { |ua| puts ua.class }

>> user\_accounts[0].user\_id # looks like ua1

>> user\_accounts[1].user\_id # should be ua2

1. If an object only contained simple fields, storing it as a row in a CSV file seems reasonable. But there are a number of problems yet to solve. For instance, how do we "unwrap" the Person objects that are part of the account\_owners array? Of course, we should wonder how safe our spreadsheet is as a persistent data storage mechanism. We won't solve that here, but the answer would be to use a database, and protect that database via backups, etc.

Let's see how YAML works to solve at least some of the problems we face. Paste the block(s) and the results as we have been doing. We will write one UserAccount object to a file using YMAL and restore that data into a new object. Notice that even the nested object Person included in account\_owners is reconstituted.

>> require 'yaml'

File.open("user\_account.txt", "w") do | file |

file.puts YAML::dump(ua1)

end

>> `cat user\_account.txt`

>> string = File.read ('user\_account.txt')

>> temp = YAML::load(string)

>> temp.class

>> temp.user\_id

>> temp.account\_owners

>> temp.account\_owners.class

>> temp.account\_owners.each { |p| puts p.class }

1. Now we can attempt to write 2 (or more, if you like) UserAccount objects to the same file and read them back out to create an Array of UserAccounts. We have to use a special string to separate one record from the next because we need every character from the YAML representation, and there are multiple lines within records. If it were one line per YAML description, a simple readlines would suffice, but that's not the case.

>> require 'yaml'

File.open("user\_accounts.txt", "w") do | file |

file.print("RECORD:") # separator

file.puts YAML::dump(ua1)

file.print("RECORD:") # separator

file.puts YAML::dump(ua2)

end

>> `cat user\_accounts.txt`

>> contents = File.read ('user\_accounts.txt')

>> raw\_data = contents.split("RECORD:")

>> raw\_data[0] # due to split

>> raw\_data[1..-1].size # 2 records

>> user\_accounts = []

raw\_data[1..-1].each do |string|

user\_accounts << YAML::load(string)

end

>> user\_accounts.each { |ua| puts ua.class }

>> user\_accounts[0].user\_id # should be ua1

>> user\_accounts[1].user\_id # should be ua2

>> user\_accounts.each { |ua| puts ua.account\_owners.class }

user\_accounts.each do |ua|

ua.account\_owners.each { |p| puts "#{p.id} #{p.last\_name}" }

end