Laboratory: Io (1)

Summary: We begin our exploration of the Io programming language.

Prerequisites: Sections 3.1 and 3.2 of Tate. Experience with Linux.

Contents:

- Preparation
- Exercises
  - Exercise 1: A Simple Method
  - Exercise 2: Containing Objects
  - Exercise 3: Static or Dynamic Typing
  - Exercise 4: Duck Typing
  - Exercise 5: Overriding
  - Exercise 6: Prototypes
- For Those with Extra Time

Preparation

a. Create a directory for the lab.

b. Verify that Io is installed on the workstation you’re using. If not, you can ssh to church, where it’s installed.

Exercises

Exercise 1: A Simple Method

a. Create an Io “class”, Person. Include slots first and last to hold the first and last name. (Yes, I know, requiring first and last names is not always appropriate for names.) Make them default values ("FIRST" for the first name, "LAST" for the last name).

b. Create an object for yourself that extends Person. Determine what your default first name and last name are. Then, replace them with your own.

c. Add a method to Person that builds a full name (first space last). You can reference the current objects with self. You can concatenate strings with .. (two periods).

d. Run this method using the Person class.
e. Run this method using your own object.

f. Make a clone of yourself and change the first name. Predict what the full name method will return. Check your answer experimentally.

**Exercise 2: Containing Objects**

a. Create an Io “class”, Course, that has one slot, instructor.

b. Create an object for this course (csc302).

c. Make yourself the instructor.

d. What do you expect the result of the following to be?

```plaintext
csc302 instructor fullname
```

e. Check your answer experimentally.

**Exercise 3: Static or Dynamic Typing**

One of the questions we asked in Ruby was whether Ruby checks the types in the body method when the method is declared or when the method is called. Let’s see what Io does.

a. We’ll start with an invalid expression. Make sure that the following is invalid.

```plaintext
3 + "hello"
```

b. Create an object, experiment.

c. Add the following method to experiment and determine whether it fails at declaration time or evaluation time.

```plaintext
experiment incorrect = method ( 3 + "hello" )
```

**Exercise 4: Duck Typing**

a. Create an objects (a clone of Object), alpha. and beta.

b. To the basic Object object, add a method of the form

```plaintext
Object doit := method ((self stuff) .. (self stuff) .. (self stuff))
```

c. What do you expect the result of the following to be?

```plaintext
Object doit
alpha doit
```
d. Check your answer, experimentally.

e. To \texttt{alpha}, add a slot named \texttt{stuff} that contains a string of your choice.

f. What do you expect the result of the following to be?

\begin{verbatim}
Object doit
alpha doit
\end{verbatim}

g. Check your answer, experimentally.

\section*{Exercise 5: Overriding}

Remember the \texttt{incorrect} method from a previous exercise? Let’s see what duck typing suggests about that approach.

a. Verify that \texttt{experiment incorrect} still fails.

b. Override the \texttt{+} method for the object 3.

\begin{verbatim}
3 + = method(x,x)
\end{verbatim}

c. What do you expect \texttt{experiment incorrect} to produce now?

d. Check your answer experimentally.

e. What do you expect \texttt{3 + 4} to return?

f. Check your answer experimentally.

g. What do you expect \texttt{4 + 3} to return?

h. Check your answer experimentally.

\section*{Exercise 6: Prototypes}

As you’ve no doubt noted, in Io inheritance is done through prototypes, which you can access with the \texttt{proto} slot. Write a method, \texttt{top}, that determines if an object is a “top level object”, one whose parent is \texttt{Object}.

\begin{verbatim}
Io> Course top
==> true
Io> csc302 top
==> false
Io> Object top
==> false
\end{verbatim}
For Those with Extra Time

Write a method that traces an Io hierarchy. (That is, your method should print the name and fields of the current object, check if the class is top level and, if not, recurse on the prototype.)