Project #4 - Getting to Know Ruby

Learning objectives

- Design and implement a complete program in an interpreted, object-oriented language.
- Employ sound principles of object-oriented program design.

Overview

In this assignment, you will become familiar with the basics of the Ruby language by duplicating the list mechanism found in Racket. Specifically, you will implement a Ruby class called *Pair* that provides much of the functionality of the cons box (a.k.a. dotted pair) in Racket.

Specifications

- Your class should be called “Pair”. Your methods should be named exactly as described below.
- You create a new pair using either the class method **`Pair.new(value1, value2)`** or using:

  the global method **`cons(value1, value2)(5 pts)`**.
- value1 and value2 can be objects of any class, including class Pair.
- Class Pair should support all of the methods listed below.
- Each of these methods is worth **5** pts if fully and correctly implemented:
  - **`car`** – return the car of the pair
  - **`cdr`** – return the cdr of the pair
  - **`to_s`** – return (not print!) a string representation of the pair. The string representation should appear exactly as it would in DrRacket. In other words, the following code:

    ```
    a = cons(1, cons(2, cons(3, cons(4, 5))))
       puts a.to_s
    ```

    should print: **`(1 2 3 4 . 5)`**

  - **`list?`** – returns true if the pair is a valid list, false if not. Should behave the same as Racket's (list?) function.
  - **`count`** – if the pair is a list, count should return the number of items in the top level of the list. If not, it should return false.
  - **`append(other)`** – if the pair is a list, append should return a new list consisting of other appended to the original list. The original list should not be modified! Other can be a list or an atomic data type. If the pair is not a list, append should return false. The new list and its values should not be linked to the original list.
  - **`null?`** – returns true only if the pair is an empty list.

- You will need to implement a null/empty list value. You may use Ruby's nil constant (an object of class NilClass), or create your own subclass of class NilClass, for this purpose. The **class method `Pair.null`** should return a null object. As in Racket a null object should return false when its pair? method is called, and true when its list? method is called(**2** pts)

- You should also implement a **`pair?`** method for ALL Ruby classes. Non-pair objects should return **false** when their **`pair?`** method is called; objects of class Pair should return true(**3** pts)

Example of Use

In the following example, indented text following the ==> symbol shows what would be printed by each puts() statement. Other text is Ruby code similar to some of the code I will use to test your implementation.

```
a = Pair.new(7, 5)
puts(a.to_s)
  ==> (7 . 5)
b = cons(1, cons(2, cons(3, Pair.null)))
puts(b.to_s)
  ==> (1 2 3)
puts(a.pair?)
  ==> true
puts(a.list?)
  ==> false
puts(b.list?)
  ==> true
puts(b.count)
```

```
  ==> 3
c = b.append(a)
puts(c)
  ==> (1 2 3 7 . 5)
somestring = "hello world"
puts somestring.pair?
  ==> false
```

What to Turn In

Submit a single, well-commented file with your Ruby code via the Pilot dropbox. The filename should be Project4.rb. Include your name in the header comments. Your code should contain only class definitions. Executing your code should produce no output.

Grading

Your submission will be based on the following criteria:

- Correctness: 45 points – Each method will be tested using a test harness that checks for correct functionality, error handling, and edge cases. Points for each method are listed above.
- Style, Readability and Documentation: 5 points –Your program should be logically organized, well-commented, and should conform to the "Coding Standards" document handed out on the first day of class (and available on the course web page under "Handouts").
