COMP 9 Lab 5: Tamagotchi!

Due: Thursday, March 17th, 12:00 PM

March 10, 2011

1 Introduction

Remember Tamagotchi? You're going to write a virtual pet program! The goal of this assignment, other than to be fun, is to give you some experience with object-oriented programming, moving a class into a separate file, and designing the parameters of a program by yourself.

2 Requirements

Many of the specifics of the game are up to you. The "virtual baby dragon" game on pages 133–138 of Pine is a good example, but your solution should be original: the actions you can take must not be identical to the dragon example, and you must keep the Pet class (or whatever you wish to call it) in a separate file from the game itself.

- Do a design sketch, as always!
- Write a *class* to represent your pet, much like the `Dragon` class Pine illustrates.
- You get to choose what kind of animal your pet is! The class name should reflect this.
- Your virtual pet should communicate with you via *standard output*. This is a fancy way of saying you'll use `puts` and `gets` rather than File I/O for this assignment.
- Write methods for each *action* you can take with your pet, such as feeding, putting to bed, taking your pet for a walk, and so on.
  - You *must* have at least five actions. This will mean at *least* five methods in your class.

  - One of these methods should be `passage_of_time` which occurs after every action is taken, and simulates the passage of time for your pet.
    * `passage_of_time` should update some *instance variables* to make your pet need to go to the bathroom, become sleepy and hungry, need exercise, and so on.
  - The actions (petting, walking, etc.) you write will also update appropriate instance variables

- The class, and thus all its methods (which are contained within the class) should live in a file called `pet.rb`
- You must also write the *game* itself; write this in the file `game.rb`
- The game should create a new pet, prompt the user to name the pet, and then loop as long as the pet is still alive, asking the user what they want to do next (and telling them what their pet has done!)
- If the user types an action that doesn't exist, your program should give an error message and prompt the user for a correct action.
- the commands you accept should include `exit` in case the user wants to quit
- It would probably be helpful to print out a list of valid commands, either each time you prompt for an action, or if the user asks for help.

2.1 Hint

Your program is going to read (via `gets`) commands from the user, and call appropriate methods based on what the user types. You can pass an arbitrary string to an object with the `send` method, but you'd better first make sure it's valid with the `respond_to?` method.

```
if pet.respond_to?(command)
    pet.send(command)
else
    # tell the user to type a valid command
end
```

Alternatively, you could just check the command against a list of actions your pet can handle.

3 Design Document

Before you write any code, write up an explanation of the following:

- What kind of animal (imaginary or otherwise) will your pet be?
- What actions can the player take? There must be at least 5.
- How will those actions affect the pet?
- What instance variables will you use to represent the *state* of your pet?
- What ends the game?

4 Handling user input

You don't have to deal with File I/O this time around; just use `gets` and `puts` to communicate with the player.

This code should all be in the `game.rb` file. The `pet.rb` file should only contain the pet class (and thus its methods).

5 Handing in your solution

When you are satisfied with your program, submit it to be graded. Note that both files will be needed:

```
provide comp9 lab5 pet.rb game.rb
```

Make sure to hand in your design sketch when you're done with it.