Regis University CC&IS
CS210 Introduction to Programming
Alice Programming Assignment 3: Methods with Parameters

This program will create an animation which uses one-shot methods and implements methods with parameters.

> ***Warning:*** The primary purpose of this assignment is to learn how to **use both procedure and function methods with parameters**. So even if your animation performs the tasks correctly, but you do not implement the scene/shot methods and function/procedure methods with parameters, you will lose significant points.

Animation Overview (details will follow later)

- Your animation will contain two objects, a Fish and a Prop.
- In scene 1, the fish will loop and say how far it looped.
- In scene 2, the fish will zig zag twice, and say how far it moved.
- In scene 3, both objects will change appearance and the fish will comment.

Program Requirements

Create an Alice program that ***minimally*** does the following:

Setting the Scene

- Start with the **seaFloor** blank slate.
- Add a Fish type object from the Swimmer class to the scene, facing front towards the camera.
- Use the properties to change the fish's height to 0.40.
- Use a one-shot methods to position the fish as follows:
  - Move the fish to the ground.
  - Move the fish up 0.5 units.
  - Move the fish right 2.0 units.
- Add an object of your choice from the Prop class to the scene, and position it in the bottom right corner of the scene.

Creating Scene/Shot Methods

- Add three empty **scene** methods (**doScene1**, **doScene2**, and **doScene3**)
- Add a new shot method (**changeAppearance**).
- Add code to **myFirstMethod** to:
  - Turn the fish left a quarter turn (to face the right side of the scene).
  - Run all three of the scene methods (**doScene1**, **doScene2**, and **doScene3**).

Creating Class Methods to be used in doScene1

- In the **Swimmer** class, create **procedure method 1** called **loopDeLoop** that causes a Swimmer object to swim in a **loop**, as follows:
  - Add one Double type **parameter** to the method, indicating the total distance to move during one loop.

    - Create a new **loop size** variable and store a quarter of the parameter value into it.
    - Perform the loop movement, as follows:
        - Move the Swimmer **up** by the **loop size**.
        - Have the Swimmer repeat 4 times:
          At the same time, **turn forward a quarter turn** and **move forward** a distance of **loop size**, with the turn and move having the same duration.
        - Finally, move the Swimmer **down** by the **loop size**.

- In the **Swimmer** class, create **function method 1** called **calcWaterPressure** to calculate and **return** the pressure on the fish in pounds per square inch, as follows:
    - Add one Double **parameter** to the method, indicating the **depth** (in feet) of the fish within the ocean.
    - Create a new variable to hold the pressure on the fish. Calculate the pressure by:
        - Air pressure at sea level is 14.7 pounds per square inch (psi).
        - To calculate water pressure, for every foot of depth the fish is underwater, add another 0.433 psi of pressure to the air pressure at sea level.
    - Return the calculated pressure.

Adding code to the doScene1 method

- Create a variable to hold the **distance to loop**.
    - Prompt the user for the distance to loop and store the user's input in the variable.
- Call the **loopDeLoop** procedure defined above to send a message to the fish object.
  Pass in the user's input (**distance to loop**) as the argument in the call, so that the fish will move the distance given when it loops all the way around.
- Create a variable to hold the **feet below sea level** where the fish is located
    - Store a random number between 1 and 100, inclusive, into the variable.
- Create a variable to hold the **water pressure on the fish**.
    - Using the **calcWaterPressure** function defined above, calculate the water pressure on the fish, with the random **feet below sea level** variable as the input argument, and store it into the **water pressure on the fish** variable.
- Have the fish say how many **feet below sea level** it is, and what pressure is being applied to it from the water.

Run the program to test **doScene1**. Debug if necessary.

Creating Class Methods to be used in doScene2

- In the **Swimmer** class, create **procedure method 2** called **zigZag** that causes a Swimmer object to swim in a zig zag motion one time (i.e. one zig and one zag), as follows:
    - Add two Double type **parameters** to the method, indicating the **forward distance** and **distance to move up/down**.
    - Perform one **zigZag** motion, as follows:

        - Simultaneously move the Swimmer **up** and **forward** by the distances specified in the parameters.
        - Simultaneously move the Swimmer **down** and **forward** by the distances specified in the parameters.

- In the **Swimmer** class, create **function method 2** called **calcZigZagDistance** to calculate and **return** the approximate total distance the swimmer swam for multiple zig zags, as follows:
    - Add three Double **parameters** to the method, indicating **forward distance**, the **up/down distance** and the **number of zigZags** performed.
    - Create a variable to hold the approximate total distance for ONE zigZag
        - Calculate the value for the variable as follows:
            - The **forward distance** plus two times the **up/down distance**.
    - Using the approximate total distance for ***one*** zigZag and the **number of zigZags** performed, calculate the total for all zigZags.
    - Return the calculated total for all zigZags.

Adding code to the doScene2 method

- Create a new variable to hold the **fish's length**
    - Get the fish's length (i.e. its depth property) and store it into the **fish's length** variable.
- Have the fish say how long it is.
- Create a new variable to hold the **up/down distance**
    - Calculate 1.5 times the fish's length and store it into the **up/down distance** variable.
- Send messages to the fish object causing it to **zigZag *twice***. The arguments to each **zigZag** call should be:
    - the **fish's length** as the forward distance
    - the **up/down distance** computed above
- Create a new variable to hold the **swim distance**
    - Use the **calcZigZagDistance** function, compute the approximate total distance the fish swam during the two zig zags and store the result in **swim distance** variable (Hint: the value 2.0 can be hardcoded as the second argument, since fish zigZagged twice).
- Have the fish say how far it swam when zig zagging.

Run the program to test **doScene2**. Debug if necessary.

Creating Methods to be used in doScene3

- Add code to the **procedure method 3**, the **changeAppearance** method to:
    - Have two **parameters**
        - A Gallery Class SModel type parameter, indicating the **object** to act upon.
        - A Double type parameter, indicating the **new opacity**.

- Send a message to change the size of the object parameter (make it bigger, your choice of how much).
- Send a message to the object parameter, using the **new opacity** parameter as the argument to the built-in **setOpacity** procedure.

Adding code to the doScene3 method

- Create a variable to hold a new opacity level.
  - Prompt the user to enter a new opacity level and store the user's answer into the variable.
    - NOTE: Tell the user that the opacity value entered must be between 0.0 and 1.0
- Simultaneously (at the same time):
  - Call the **changeAppearance** procedure defined above with the **fish** object as the first argument and new opacity level entered by the user as the second argument.
  - Call the **changeAppearance** procedure defined above with the **prop** object as the first argument and new opacity level entered by the user as the second argument.
- Have the fish think something about what just happened.

Run the program to test **doScene3**. Debug if necessary.

Final testing

- Run, test, and debug your Alice program, until it works correctly.
- Make sure the duration of each movement in your animation is long enough to view comfortably.

Extra credit* (5 pts)

*Save a copy of the normal part of the assignment first, and then save it under a new name before starting on the extra credit, in case your efforts on the extra credit are unsuccessful.

Optionally you may add to the animation for extra credit, as follows:

- ***Within*** one of the object ***classes***, create a **new** class **procedure** or **function** method that requires at least one parameter.
- Within the method, causes an object of that class to perform at least one new behavior, implemented using the **parameter** value(s).

  NOTE: The new behavior cannot be a behavior implemented by any of the course materials, or be a behavior already implemented in this program or a previous program.

Your new method should be called from an additional **extraCreditScene** method.

Warning: Do not attempt to implement the extra credit unless you have first successfully implemented the normal part of the assignment. ***You will not receive any extra credit if the normal part of the assignment is not functioning properly.***

Submission

This programming assignment is due by midnight of the date listed on the Assignments page.

Submit your program source code (the **.a3p** file) to the **Alice Prog Assn 3** submission folder (located under the **Assignments/Dropbox** tab in the online course).

Before submitting your program file, you MUST re-name it as follows:

`LastnameAliceAssn3.a3p`

For example: **`SmithAliceAssn3.a3p`**

Grading

The rubric that will be used to grade your program is linked on the same assignment page that you downloaded this file from.

WARNING:

*Programs submitted more than 5 days past the due date will **not** be accepted, and will receive a grade of 0.*