TY CSE AY-2023-24 Sem-II

iOS Lab (6CS381)

Assignment No 8

Due date- 20/03/2024

1. Create a `Spaceship` class with three variable properties: `name`, `health`, and `position`. The default value of `name` should be an empty string and `health` should be 0. `position` will be represented by an `Int` where negative numbers place the ship further to the left and positive numbers place the ship further to the right. The default value of `position` should be 0.

2. Create a `let` constant called `falcon` and assign it to an instance of `Spaceship`. After initialization, set `name` to "Falcon."

3. Go back and add a method called `moveLeft()` to the definition of `Spaceship`. This method should adjust the position of the spaceship to the left by one. Add a similar method called `moveRight()` that moves the spaceship to the right. Once these methods exist, use them to move `falcon` to the left twice and to the right once. Print the new position of `falcon` after each change in position.

4. The last thing `Spaceship` needs for this example is a method to handle what happens if the ship gets hit. Go back and add a method `wasHit()` to `Spaceship` that will decrement the ship's health by 5, then if `health` is less than or equal to 0 will print "Sorry, your ship was hit one too many times. Do you want to play again?" Once this method exists, call it on `falcon` and print out the value of `health`.

5. Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system and can only move left and right while obstacles "fall" from top to bottom. Throughout the exercises, you'll create classes to represent different types of spaceships that can be used in the game. The base class `Spaceship` has been provided for you below.

```
class Spaceship {
    var name: String = ""
    var health = 100
    var position = 0

    func moveLeft() {
        position -= 1
    }
    func moveRight() {
        position += 1
    }

    func wasHit() {
        health -= 5
        if health <= 0 {
            print("Sorry, your ship was hit one too many times. Do you want to play again?")
        }
    }
}
```

6. Define a new class `Fighter` that inherits from `Spaceship`. Add a variable property `weapon` that defaults to an empty string and a variable property `remainingFirePower` that defaults to 5.

7. Create a new instance of `Fighter` called `destroyer`. A `Fighter` will be able to shoot incoming objects to avoid colliding with them. After initialization, set `weapon` to "Laser" and `remainingFirePower` to 10. Note that since `Fighter` inherits from `Spaceship`, it also has properties for `name`, `health`, and `position`, and has methods for `moveLeft()`, `moveRight()`, and `wasHit()` even though you did not specifically add them to the declaration of `Fighter`. Knowing that, set `name` to "Destroyer," print `position`, then call `moveRight()` and print `position` again.

8. Try to print `weapon` on `falcon`. Why doesn't this work? Provide your answer in a comment or a print statement below, and remove any code you added that doesn't compile.

9. Add a method to `fighter` called `fire()`. This should check to see if `remainingFirePower` is greater than 0, and if so, should decrement `remainingFirePower` by one. If `remainingFirePower` is not greater than 0, print "You have no more fire power." Call `fire()` on `destroyer` a few times and print `remainingFirePower` after each method call.

10. Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system and can only move left and right while obstacles "fall" from top to bottom. Throughout the exercises, you'll create classes to represent different types of spaceships that can be used in the game. The base class `Spaceship` and one subclass `Fighter` have been provided for you below.

```
class Spaceship {
    var name: String = ""
    var health = 100
    var position = 0
    func moveLeft() {
        position -= 1
    }

    func moveRight() {
        position += 1
    }

    func wasHit() {
        health -= 5
        if health <= 0 {
            print("Sorry, your ship was hit one too many times. Do you want to play again?")
        }
    }
}

class Fighter: Spaceship {
    var weapon = ""
    var remainingFirePower = 5

    func fire() {
        if remainingFirePower > 0 {
            remainingFirePower -= 1
        } else {
            print("You have no more fire power.")
        }
    }
}
```

11. Define a new class `ShieldedShip` that inherits from `Fighter`. Add a variable property `shieldStrength` that defaults to 25. Create a new instance of `ShieldedShip` called `defender`. Set `name` to "Defender" and `weapon` to "Cannon." Call `moveRight()` and print `position`, then call `fire()` and print `remainingFirePower`.

12. Go back to your declaration of `ShieldedShip` and override `wasHit()`. In the body of the method, check to see if `shieldStrength` is greater than 0. If it is, decrement `shieldStrength` by 5. Otherwise, decrement `health` by 5. Call `wasHit()` on `defender` and print `shieldStrength` and `health`.

13. When `shieldStrength` is 0, all `wasHit()` does is decrement `health` by 5. That's exactly what the implementation of `wasHit()` on `Spaceship` does! Instead of rewriting that, you can call through to the superclass implementation of `wasHit()`. Go back to your implementation of `wasHit()` on `ShieldedShip` and remove the code where you decrement `health` by 5 and replace it with a call to the superclass's implementation of the method. Call `wasHit()` on `defender`, then print `shieldStrength` and `health`.

14. Note: The exercises below are based on a game where a spaceship avoids obstacles in space. The ship is positioned at the bottom of a coordinate system and can only move left and right while obstacles "fall" from top to bottom. The base class `Spaceship` and subclasses `Fighter` and `ShieldedShip` have been provided for you below. You will use these to complete the exercises.

```
class Spaceship {
    let name: String
    var health: Int
    var position: Int

    func moveLeft() {
        position -= 1
    }

    func moveRight() {
        position += 1
    }

    func wasHit() {
        health -= 5
        if health <= 0 {
            print("Sorry, your ship was hit one too many times. Do you want to play again?")
        }
    }
}

class Fighter: Spaceship {
    let weapon: String

    var remainingFirePower: Int


    func fire() {
        if remainingFirePower > 0 {
            remainingFirePower -= 1
        } else {
            print("You have no more fire power.")
        }
    }
}

class ShieldedShip: Fighter {
   var shieldStrength: Int

    override func wasHit() {
       if shieldStrength > 0 {
          shieldStrength -= 5
       } else {
          super.wasHit()
       }
    }
}
```

15. Note that each class above has an error by the class declaration that says "Class has no initializers." Unlike structs, classes do not come with memberwise initializers because the standard memberwise initializers don't always play nicely with inheritance. You can get rid of the error by providing default values for everything, but it is common, and better practice, to simply write your own initializer. Go to the declaration of `Spaceship` and add an initializer that takes in an argument for each property on `Spaceship` and sets the properties accordingly.

16. Then create an instance of `Spaceship` below called `falcon`. Use the memberwise initializer you just created. The ship's name should be "Falcon."

17. Writing initializers for subclasses can get tricky. Your initializer needs to not only set the properties declared on the subclass, but also set all of the uninitialized properties on classes that it inherits from. Go to the declaration of `Fighter` and write an initializer that takes an argument for each property on `Fighter` and for each property on `Spaceship`. Set the properties accordingly. (Hint: you can call through to a superclass's initializer with `super.init` *after* you initialize all of the properties on the subclass).

18. Then create an instance of `Fighter` below called `destroyer`. Use the memberwise initializer you just created. The ship's name should be "Destroyer."

19. Now go add an initializer to `ShieldedShip` that takes an argument for each property on `ShieldedShip`, `Fighter`, and `Spaceship`, and sets the properties accordingly. Remember that you can call through to the initializer on `Fighter` using `super.init`.

20. Then create an instance of `ShieldedShip` below called `defender`. Use the memberwise initializer you just created. The ship's name should be "Defender."

21. Create a new constant named `sameShip` and set it equal to `falcon`. Print out the position of `sameShip` and `falcon`, then call `moveLeft()` on `sameShip` and print out the position of `sameShip` and `falcon` again. Did both positions change? Why? If both were structs instead of classes, would it be the same? Why or why not? Provide your answer in a comment or print statement below.

22. Assume you are an event coordinator for a community charity event and are keeping a list of who has registered. Create a variable `registrationList` that will hold strings. It should be empty after initialization.

23. Your friend Sara is the first to register for the event. Add her name to `registrationList` using the `append(_:)` method. Print the contents of the collection.

24. Add four additional names into the array using the `+=` operator. All of the names should be added in one step. Print the contents of the collection.

25. Use the `insert(_:at:)` method to add `Charlie` into the array as the second element. Print the contents of the collection.

26. Somebody had a conflict and decided to transfer registration to someone else. Use array subscripting to change the sixth element to `Rebecca`. Print the contents of the collection.

27. Call `removeLast()` on `registrationList`. If done correctly, this should remove `Rebecca` from the collection. Store the result of `removeLast()` into a new constant `deletedItem`, then print `deletedItem`.

28. These exercises reinforce Swift concepts in the context of a fitness tracking app. Your fitness tracking app shows users a list of possible challenges, grouped by activity type (i.e. walking challenges, running challenges, calisthenics challenges, weightlifting challenges, etc.) A challenge could be as simple as "Walk 3 miles a day" or as intense as "Run 5 times a week."

29. Using arrays of type `String`, create at least two lists, one for walking challenges, and one for running challenges. Each should have at least two challenges and should be initialized using an array literal. Feel free to create more lists for different activities.

30. In your app you want to show all of these lists on the same screen grouped into sections. Create a `challenges` array that holds each of the lists you have created (it will be an array of arrays). Using `challenges`, print the first element in the second challenge list.

31. All of the challenges will reset at the end of the month. Use the `removeAll` to remove everything from `challenges`. Print `challenges`.

32. Create a new array of type `String` that will represent challenges a user has committed to instead of available challenges. It can be an empty array or have a few items in it.

33. Write an if statement that will use `isEmpty` to check if there is anything in the array. If there is not, print a statement asking the user to commit to a challenge. Add an else-if statement that will print "The challenge you have chosen is <INSERT CHOSEN CHALLENGE>" if the array count is exactly 1. Then add an else statement that will print "You have chosen multiple challenges."

34. Create a variable `[String: Int]` dictionary that can be used to look up the number of days in a particular month. Use a dictionary literal to initialize it with January, February, and March. January contains 31 days, February has 28, and March has 31. Print the dictionary.

35. Using subscripting syntax to add April to the collection with a value of 30. Print the dictionary.

36. It's a leap year! Update the number of days in February to 29 using the `updateValue(_:, forKey:)` method. Print the dictionary.

37. Use if-let syntax to retrieve the number of days under "January." If the value is there, print "January has 31 days", where 31 is the value retrieved from the dictionary.

38. Given the following arrays, create a new [String : [String]] dictionary. `shapesArray` should use the key "Shapes" and `colorsArray` should use the key "Colors." Print the resulting dictionary.

39. Print the last element of `colorsArray`, accessing it through the dictionary you've created. You'll have to use if-let syntax or the force unwrap operator to unwrap what is returned from the dictionary before you can access an element of the array.

40. These exercises reinforce Swift concepts in the context of a fitness tracking app. In previous app exercises you've written code to help users with run pacing. You decide that you could use a dictionary to let users store different paces that they regularly run at or do interval training with.

41. Create a dictionary `paces` of type [String: Double] and assign it a dictionary literal with "Easy", "Medium", and "Fast" keys corresponding to values of 10.0, 8.0, and 6.0. These numbers correspond to mile pace in minutes. Print the dictionary.

42. Add a new key/value pair to the dictionary. The key should be "Sprint" and the value should be 4.0. Print the dictionary.

43. Imagine the user in question gets faster over time and decides to update his/her pacing on runs. Update the values of "Medium" and "Fast" to 7.5 and 5.8, respectively. Print the dictionary.

44. Imagine the user in question decides not to store "Sprint" as one his/her regular paces. Remove "Sprint" from the dictionary. Print the dictionary.

45. When a user chooses a pace, you want the app to print a statement stating that it will keep him/her on pace. Imagine a user chooses "Medium." Accessing the value from the dictionary, print a statement saying "Okay! I'll keep you at a <INSERT PACE VALUE HERE> minute mile pace."

****