Module 4
Swift Development

Assignment 5 - MADP401- Wednesday Sep 11, 2019

Due: Monday September 16, 6:30 pm:

Submission: Please create a pull-request to your branch on upstream

Swift 5 Inheritance + Error + Extension:

Problem 1: Shape and Inherited Computed Properties

Design a class called Shape which is the parent of the classes Rectangle, Square and Circle. Also define classes for Rectangle, Square and Circle to be child of the class Shape.
The class shape has area and circumference.

- Define area and circumference to be computed properties of the class Shape.
- Use override to override the computed properties of the classes Rectangle, Square and Circle.
- Provide some example to show your implementation works.

Problem 2: Inheritance

a) Define a Swift class called Product which is a base class.
b) Add the following instance stored properties to this class:

Properties	Type
productID	Integer
productName	String
productPrice	Double
productMadeInCountry	String

c) Define and implement an initializer for this class as following

```
func init(productid: Int, productName: String, productPrice: Double,
productMadeInCountry: String) {
  self. productID = productid
  self. productName = productName
  self. productPrice = productPrice
  self. productMadeInCountry = productMadeInCountry
}
```

d) In playground create one object from the class Product with the following values for the properties:

Properties	Type
ProductID	110
productName	“Diet Pepsi”

productPrice	2
productMadeInCountry	USA

e) Define a Swift class for each of the following entities using the Product class you have defined in Problem 1. (Use inheritance!)

Class: Drink	**Type**
drinkID	Int
drinkName	String
drinkPrice	Double
drinkMadeInCountry	String
isDrinkDiet	Boolean
drinkSize	Int

Class: Food	**Type**
foodID	Int
foodName	String
foodPrice	Double
foodMadeInCountry	String
foodCalorie	Int
foodSize	Int
foodIngredients	Array of String

Class: Cloth	**Type**
ClothID	Int
ClothName	String
ClothPrice	Double
ClothMadeInCountry	String
ClothMaterials	Array of Material

Class: Material	**Type**
MaterialCode	Int
MaterialName	String

f) Define and implement one initializer for each of the above classes.
g) Define the properties of the classes.
h) In the playground class create one object from each of these classes with some arbitrary values.

i) Define a Swift class called ShoppingCart. The shoppingCart class works like a container for your shopping. Imagine you have bought the following items:

Drinks	**Amount: 3**
drinkID	412
drinkName	Pepsi
drinkPrice	2$
drinkMadeInCountry	USA
isDrinkDiet	NO
drinkSize	150

Drinks	**Amount: 1**
drinkID	183
drinkName	Ginger Zero
drinkPrice	3$
drinkMadeInCountry	Canada
isDrinkDiet	Yes
drinkSize	200

Food	**Amount: 2**
foodID	100
foodName	Chicken
foodPrice	8$/lb
foodMadeInCountry	Canada
foodCalorie	350
foodSize	4lb
foodIngredients	chicken, oil, chees

Food	**Amount: 2**
foodID	101
foodName	Pasta
foodPrice	18$ / lb
foodMadeInCountry	Canada
foodCalorie	250
foodSize	3lb
foodIngredients	pasta, meet, spinach

Cloth	**Amount: 1**

ClothID	701
ClothName	T-shirt
ClothPrice	15$
ClothMadeInCountry	China
ClothMaterials	cotton (code: 10), Nylon (code: 11)

j) Define appropriate states for the ShoppingCart class.
k) Define (specify the signature of the method) and implement the following methods to the class ShoppingCart:
l) A method using which you can add your purchases items (listed above) to your shopping cart.
m) A method using which you can calculate the total amount you need to pay for your entire purchase.
n) A method using which you can print just the name of the items you have purchases.
o) Now in the playground, create an instance (object) from class ShoppingCart with some arbitrary values.
p) Call all methods you defined above in the ShoppingCart and with some sample inputs.

Problem 3: Extensions

Write two extensions for String

- Extension1:
  - Add an instance method which checks whether the string is palindrome or not. The signature is: `func isPalindrome () -> Bool`
  - Add an instance method which returns the reverse of the string `func reverseIt()-> String`

- Extension2:
  - Add an instance computed property which represents number of digits in the string
  - add an instance computed property which represents the uppercase version of the string

Problem 4: Error Handling + Extension

Using an enum called StringError define the following 3 errors:

- InvalidStringFormatForIntConversionError
- InvalidEmailFormatError
- NotValidPassword(numberNeeded: Int)

Write an extension for String and add the following methods:

- A method which converts the String to an Int if possible. The method will throws an error called InvalidStringFormatForIntConversion if it is not possible to convert the String to an integer. Only a string whose all characters are digits is allowed to be converted to an Int, otherwise it cannot be converted.
- A method which checks whether the String has a valid email format. A valid email format is like aaaa@b.c. If the string does not have a valid email format, the method will returns an error called InvalidEmailFormatError
- A method which checks wither the string is long enough to be a valid password. A valid password is at least 8 characters long. If the string is less than 8 characters, then the method will throw an error and indicates how many more characters is needed in order to have a valid password. For instance if the string has 6 characters, then it needs 2 more characters in order to be a valid password.

Design a class called TestExtension. The class has 3 static methods each of which test the above scenarios and handle the errors using try/catch block. The methods are

- `func convertStringToInt (word: String) -> Int?`
- `func isItAnEmail(word: String)-> Boolean`
- `func validatePassword(password: String)-> Boolean`

Write some tests in playground to show how the above classes works.