Examen para desarrolladores IOs

Examen para desarrollador SR IOs

Pregunta 1: Considere el siguiente código y responda

- ¿Dónde está el error?

  En la llamada a la función printString esta se llama sin nombre de parámetro

- ¿Qué causa este error?

  Esto ocasiona que el se este buscando una función que recibe un parámetro sin etiqueta

- ¿Cuál es la forma correcta de solucionarlo?

  Se puede ajustar el llamado para que concuerde con la función

  Ejemp: printString(string: userPref)

```
var defaults = NSUserDefaults.standardUserDefaults ()
 var userPref = defaults.stringForKey ("userPref")!
 printString (userPref)

 func printString (string: String) {
 println (cadena)
 }
```

Pregunta 2: El siguiente fragmento de código da como resultado un error de tiempo de compilación

- Explica por qué ocurre un error en tiempo de compilación.

  Primero las variables se declaran de la forma palabra reservada var seguido del nombre y tipo si esta no tiene un tipo se infiera de instanciarla

  Segundo en la función add esta trata de agregar un elemento a un array llamado ítems

- ¿Cómo puedes arreglarlo?

  Primero se tiene que invertir la forma de la declaración del array elementos

  var elementos = [Int] ()

  Segundo el ítems no existe en ese contexto en todo caso se puede almacenar el valor en el array elementos ya que es de tipo Int

  elementos.append(x)

```
struct IntStack {
 elementos var = [Int] ()
 func add (x: Int) {
 items.append (x) // Error de tiempo de compilación aquí.
 }
 }
```

Pregunta 3: ¿Cómo convertir Swift String en una matriz?

Con una función map para cada uno de los elementos del string

let str = “ABC”

let array = str.map { String($0) }

Pregunta 4: Considere el siguiente código y responda:

- ¿Cuál es el valor de la variable `len` y por qué?

  El valor de len es 5, primero se hacen operaciones con objetos llamados matriz2 y y matiz1 los cuales no se relacionan con array1, después se agrega un 6 a un array2 que tampoco esta relacionado, en caso de que estuviera asignado el valor de array uno a otro array este genera un copia por lo cual no afecta.

```
var array1 = [1, 2, 3, 4, 5]
 var matriz2 = matriz1
 array2.append (6)
 var len = array1.count
```

Pregunta: 5 Considere el siguiente código y responda:

- ¿Dónde está el error y por qué?

  El error es en la ultima línea ya que se intentan sumar constantes de diferentes tipos, se necesita ser del mismo tipo para se puedan sumar aparte que la palabra reservada var esta después del nombre de la variable

- ¿Cómo se puede arreglar?

  Conviertiendo op1 y op2 a Double para que sean del mismo tipo y que no se pierda información de op3

  var resultado = Double(op1) + Double(op2) + op3

```
let op1: Int = 1
 let op2: UInt = 2
 let op3: Double = 3.34
 resultado var = op1 + op2 + op3
```

Pregunta 6: Swift define el alias de tipo `AnyObject` para representar instancias de cualquier tipo de referencia, y se define internamente como un protocolo.

protocol optimissa { associatedtype : AnyObject}

Considere el siguiente código:

```
var matriz = [AnyObject] ()
 Prueba de estructura {}
 array.append (Prueba ())
```

Este código genera un error de compilación, con el siguiente mensaje de error:

```
El tipo 'Prueba' no se ajusta al protocolo 'AnyObject'
```

El fallo es obvio porque una estructura es un valor y no un tipo de referencia y como tal, no se implementa y no se puede convertir al protocolo `AnyObject`.

Ahora considere el siguiente código:

```
var matriz = [AnyObject] ()
 array.append (1)
 array.append (2.0)
 array.append ("3")
 array.append ([4, 5, 6])
 array.append ([7: "7", 8: "8"])

 Prueba de estructura {}
 array.append (Prueba ())
```

La `array` matriz se completa con valores de tipo respectivamente int, double, string, array y dictionary. Todos ellos son tipos de valor y no tipos de referencia, y en todos los casos el compilador no informa de ningún error. **¿Por qué?**

El problema es que una estructura no se puede ajustar al protocolo anyobject

Ejercicio

Juego de las siete y media

Se propone implementar el conocido juego de cartas de "las 7 y media".

Para simplificar, solo habrá dos jugadores: el usuario contra la máquina que hace de banca. El usuario va pidiendo cartas una a una hasta que se pasa de 7 y medio o decide plantarse. La máquina no va sacando cartas una a una sino que obtiene una puntuación generada al azar.

Para crear el proyecto, elige la plantilla de App. En la segunda pantalla del asistente dale como nombre SieteyMedia y asegúrate que en el interface pone Storyboard (como todas las apps que hemos hecho hasta ahora lo usan ya debería salir por defecto).

Estructura de clases del modelo

Necesitáis implementar primero el enum Palo y las clases Carta y Mano. Estos se proponían como ejercicio el primer día de clase, aunque es posible que no os haya dado tiempo a hacerlos, podéis hacerlos ahora.

Además hay que añadir dos clases necesarias para poder jugar: la Baraja y el propio Juego

Clase `Baraja`¶

Todas las cartas de la baraja. Del 1 al 12 de los cuatro palos, menos 8 y 9

Propiedades: cartas, un array de Carta

Métodos:

El init() debe rellenar el array de cartas con todas las cartas de la baraja. Podéis ir generando todos los números de todos los palos con un bucle doble de este estilo:

repartirCarta(): devuelve la última carta de la baraja y la elimina de ella. Es lo que hace exactamente el método de la clase Array popLast(), devolver el último valor de un array y eliminarlo de él.

barajar(): debe cambiar al azar el orden de las cartas en el Array. Esto lo hace directamente el método shuffle() del Array.

Clase `Juego`¶

Es la clase que implementa las reglas del juego de las siete y media. Os dejo aquí una implementación que podéis usar para no gastar demasiado tiempo haciendo la vuestra propia desde cero. No obstante no está completa, tenéis que implementar el método acabarPartida().

Para simplificar el juego, en esta implementación la máquina no saca cartas de verdad, una a una. Se genera una puntuación al azar entre 1 y 7.5 y se informa al usuario de la puntuación que ha sacado. Para que el juego así tenga sentido, primero juega el jugador humano y luego la máquina.

Interfaz simplificada

El ViewController contendrá una instancia de la clase Juego.

En esta versión muy simplificada de la interfaz solo aparecen en pantalla tres botones: "pedir carta", "plantarse" y "nueva partida", pero no se ven las cartas gráficamente. Eso sí, el juego debería funcionar correctamente, imprimiendo los mensajes con print.

Tendrás que conectar con action (gráficamente) los botones del juego con funciones del view controller que a su vez llamen a los métodos del objeto juego.

Interfaz completa

En la interfaz completa deberían aparecer pintadas las cartas en pantalla conforme se van repartiendo. Además, los botones se deberían habilitar/deshabilitar adecuadamente (por ejemplo, si la partida se ha terminado no se puede pedir carta).

Los entregables a evaluar serán:

- Paquete de código, o url de git con la solución
- Hoja con las respuestas a las preguntas.