init methods / more on optionals / connecting UI to code

CS112 Unit 8

Max Luttrell, Fall 2016

init methods

- we can use **init()** methods in a class to **initialize** properties in a new instance of the class
- note: after an init() function is complete, each property must be initialized

Player class

```
class Player {
    var name = ""
    var weight = 0.0
    var height = 0.0
    var age = 0
    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
    func incrementAge() {
        age += 1
    }
}
```

```
var qb = Player()
qb.name = "Joe Montana"
qb.weight = 93
qb.height = 1.88
qb.age = 60
qb.printInfo()
```

sample debug output

Joe Montana
93.0 kg, 1.88 m, 60 yrs

Player init()

```
class Player {
    var name = ""
    var weight = 0.0
    var height = 0.0
    var age = 0
    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
    func incrementAge() {
        age += 1
    }
    init(name: String, weight: Double, height: Double, age: Int) {
        self.name = name
        self.weight = weight
        self.height = height
        self.age = age
    }
}
```

arg labels

```
var qb = Player(name: "Joe Montana", weight: 93, height: 1.88, age: 60)
qb.printInfo()
```

sample debug output

Joe Montana
93.0 kg, 1.88 m, 60 yrs

Player init()

```
class Player {
    var name = ""
    var weight = 0.0
    var height = 0.0
    var age = 0
    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
    func incrementAge() {
        age += 1
    }
    init(name: String, weight: Double, height: Double, age: Int) {
        self.name = name
        self.weight = weight
        self.height = height
        self.age = age
    }
    init(name: String) {
        self.name = name
    }
}
```

```
var wr = Player(name: "Jerry Rice")
wr.printInfo()
```

sample debug output

Jerry Rice
0.0 kg, 0.0 m, 0 yrs

optionals

- Recall: an optional is like a variable, but might not hold a value (i.e. **nil**)

```
var firstName: String = "John"
var lastName: String = "Adams"
var middleName: String? = "Quincy"

middleName = nil

// now, middleName no longer contains any value
```

- we use a question mark to indicate an optional

implicitly unwrapped optionals

- Sometimes, we will have a property that must be optional, but we know that it will quickly get a value (quickly enough that we'll never access it before it gets the value)

- It would be nice to be able to use such optionals without needing to force-unwrap each time

- Swift provides an **implicitly unwrapped optional** to deal with this situation. We declare it with an **exclamation point**

example

- example: we have two classes: Country, and City
- each country has a City named capitalCity and each City has a Country named country

```
class Country {
    let name: String
    var capitalCity: City!
    init(name: String, capitalName: String) {
        self.name = name
        self.capitalCity = City(name: capitalName, country: self)
    }
}

class City {
    let name: String
    unowned let country: Country
    init(name: String, country: Country) {
        self.name = name
        self.country = country
    }
}
```

implicitly unwrapped optionals - why?

- in general, regular optionals can be used
- iOS frameworks use implicitly unwrapped optionals frequently in class initialization

Exercise 7A

- in a Swift playground create the Player class below. Notice that you get an error! think about why.

- make all of the properties optionals. do you still get the error?

- change the properties back to regular variables (not optionals). add an init() method to the function which can set all the properties

- create an object using your init method, giving it some reasonable values

- call printInfo() on your object

```
class Player {
    var name: String
    var weight: Int
    var height: Int
    var age: Int

    func printInfo() {
        print("\(name)")
        print("\(weight) kg, \(height) m, \(age) yrs")
    }
}
```

@IBAction and @IBOutlet attributes

- in order to connect our Swift code to UI elements in Storyboard (Interface Builder), we can use the following **attributes**
  - **`@IBOutlet`** - property which can reference an object in Storyboard (e.g. a Label)
  - **`@IBAction`** - code to perform upon an event in Storyboard (e.g. user taps a button)

```
@IBOutlet weak var resultLabel: UILabel!

@IBAction func Giants(sender: UIButton) {
    dismissViewControllerAnimated(true, completion: nil)
}
```

demo - build an app with outlets and actions

Exercise 7B

- we have built an app which has a button and a label. the “Go” button increments num, which we then display on the label
- add another button to your project. if the user taps this button, decrement num and update the label with the new value.