2) Swift: Classes, Structures, Enumerations

1) Mensa App
An app is supposed to be developed that can display available canteens on a campus that are available for students and academic employees. Typically, canteens have a name and an address to find them and offer different meals per day. A meal also usually is made up of a name and a price (and some ingredients of course, but this information is not required for now). Prices usually vary and depend on the person buying a meal. The price for a meal is usually cheaper than for employees or guests. For this task, only an empty Playground-file is required.

1) Create types for a Canteen and Meal with the respective properties mentioned in the text.
   ```swift
   struct Canteen {
     let name: String
     let address: String
   }
   
   struct Meal {
     let name: String
     let price: Double
   }
   ```

2) Per meal only one price is supposed to be stored. Other prices are to be calculated based on this ‘base price’. The ‘base price’ is always the student price. Create two more price properties for employees and externals that depend on the student price. Employees pay 30% more, externals 75%.

   ```swift
   struct Canteen {
     let name: String
     let address: String
   }
   
   struct Meal {
     let name: String
     let studentPrice: Double
     var category: String? // 1.3
     
     var employeePrice: Double {
       get {
         return studentPrice + (studentPrice * 0.3)
       }
     }
     
     var externalsPrice: Double {
       get {
         return studentPrice + (studentPrice * 0.75)
       }
     }
   }
   ```

3) Some canteens categorize their meals, e.g. MensaVital, but such an information is not mandatory. Add a property to the meal that reflects this requirement and make a shorthand initializer available that only requires a meal's name and price.
3) Consider the following code snippet:

```swift
class Person {
    let name: String

    init(name: String) {
        self.name = name
    }
}

class Student: Person {
    let studentID: String
    var faculty: String

    init(name: String, faculty: String, studentID: String) {
        self.studentID = studentID
        super.init(name: name)
        self.faculty = faculty
    }

    convenience init(name: String, faculty: String) {
        self.init(name: name, faculty: faculty, studentID: "RandomString")
    }
}
```

Because all properties of the class must be initialized before the initialization process if forwarded to the super class. The rule is violated by initializing faculty after the call to super.

2) Implement the designated initializer of the Person class and the convenience initializer of the Student class from the code snippet of task 2.1.

3) Consider the following code snippet:
struct Point {
    let x = 0.0, y = 0.0
}

class GeoStructure {
    let origin: Point
    let height, width: Double

    init(withOrigin origin: Point, height: Double, andWidth width: Double) {
        self.origin = origin
        self.height = height
        self.width = width
    }

    init(withGeoStructure structure: GeoStructure) {
        origin = structure.origin
        height = structure.height
        width = structure.width
    }

    convenience init(withOrigin origin: Point) {
        self.init(withOrigin: origin, height: 0.0, andWidth: 0.0)
    }
}

class SubGeoStructure: GeoStructure {
    let angle = 0.0
}

1) What initializers can be used to create a new instance of SubGeoStructure? List them all.
All designated and convenience initializers of GeoStructure. SubGeoStructure does not provide own initializers.
2) You add a designated initializer to SubGeoStructure. What initializers can now be used to create a new instance of SubGeoStructure? List them all.
Only the added designated initializer is available, because initializers are not inherited from superclass if own initializers are provided.
1) What do you have to do, to get the set of initializers from 3.1) + the newly added initializer?
Make the designated initializer a convenience initializer (Rule 1) or override all initializers from the superclass (Rule 2)
3) What initializers of GeoStructure cannot be called by forwarding the initialization from a designated initializer of SubGeoStructure? List them all.
The convenience initializers due to 2-phase-initialization rules.
4) To which initializers of GeoStructure and SubGeoStructure can a convenience initializer of SubGeoStructure forward the initialization process? Assume SubGeoStructure offers a designated initializer itself. List them all.
To the assumed designated initializer and other convenience initializers.
3) Value-Type Initialization
1) Assume stored properties of a value-type, e.g. a struct, to be optionals. Extend the given table for the generation of default and member wise initializers for the case of mutable and immutable stored properties which are optionals. What could be reasons for the generation of default and member wise constructors?

<table>
<thead>
<tr>
<th>Given</th>
<th>Default Initializer</th>
<th>Memberwise Initializer</th>
</tr>
</thead>
<tbody>
<tr>
<td>mutable stored properties</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>with default value</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>no default value</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
optional properties in Swift are not allowed to be immutable, because reassigning the constant to a value different from nil or to nil, respectively, would not be allowed.

- since Optionals are also valid types when containing a nil-value, default and member wise initializers are generated in any case.
- default and member wise initializer are automatically generated to promote the safe initialization process of value- and reference-types while reducing the amount of “boilerplate” code the developer has to write in order to comply to the initialization rules.

4) Optionals

1) Consider the method implementation in the following code snippet. Why would this code not compile?

```swift
func httpRequestWithURL(url: NSURL, andHTTPHeaders headers: Dictionary<String, String>?) -> NSURLRequest {
    let urlRequest = NSMutableURLRequest(URL: url)
    for (header, value) in headers {
        urlRequest.addValue(value, forHTTPHeaderField: header)
    }
    return urlRequest
}
```

*Because the optional is accessed without having it unwrapped first. *Access of unwrapped headers-optional*

2) Add exactly one character in the code snippet above to make compilation of the code possible. Is this code safe to execute in any case? If not, name a method usage that is valid, but would break the code.

```swift
func httpRequestWithURL(url: NSURL, andHTTPHeaders headers: Dictionary<String, String>?) -> NSURLRequest {
    let urlRequest = NSMutableURLRequest(URL: url)
    for (header, value) in headers! {
        urlRequest.addValue(value, forHTTPHeaderField: header)
    }
    return urlRequest
}
```

*Exclamation point was added in for-declaration to forcefully unwrap the headers-optional. This code can crash at runtime if headers is nil.*

3) Change the code snippet from the second task so that it compiles and is safe to execute in any case.
func httpRequestWithURL(url: NSURL, andHTTPHeaders headers: Dictionary<String, String>?) -> NSURLRequest {
    let urlRequest = NSMutableURLRequest(URL: url)
    if let unwrappedHeaders = headers {
        for (header, value) in unwrappedHeaders {
            urlRequest.addValue(value, forHTTPHeaderField: header)
        }
    }
    return urlRequest
}