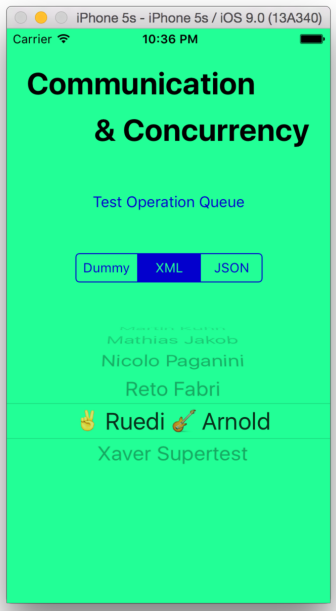
Übung 6: Kommunikation & Nebenläufigkeit

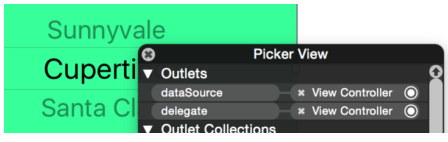


Diese Übung hat als ersten Schwerpunkt Kommunikation über das HTTP-Protokoll. Konkret werden mittels blockierender Framework-Funktionen XML- und JSON-Dateien via HTTP-Anfragen von einem Server geholt, geparst und in einer PickerView dargestellt. Die Auswahl der Datenquelle geschieht dabei über ein SegementedControl. Sie lernen dabei ebenfalls die von Apple zur Verfügung gestellten Mechanismen zum Parsen von XML- und JSON-Dateien kennen. Im weiteren lernen sie mit Operation-Queues und Operations die beiden Alternativen von Apple zur Verwendung von Threads unter iOS kennen. Insbesondere implementieren sie dabei eigene einfache Closures.

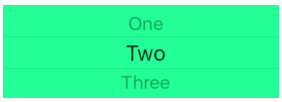
1. **Neues Projekt "ComAndCon"**
   Erstellen sie in Xcode ein neues iPhone-Projekt, wählen sie "Single-View Application", Produktname "ComAndCon". Erstellen sie im Interface Builder das Layout für die App anhand des rechts oben vorgegebenen Screenshots. Im Wesentlichen braucht die App also einen Titel, einen Knopf "Test Operation Queue", einen Auswahlknopf (SegementedControl) mit 3 Möglichkeiten und eine PickerView. Den gesamten Code dieser Aufgabe implementieren sie in der ViewController-Klasse. Diese Klasse soll am Schluss folgende Deklarationen haben:

```
class ViewController: UIViewController, UIPickerViewDataSource,
  UIPickerViewDelegate, XMLParserDelegate {

  @IBOutlet weak var pickerView: UIPickerView!

  let dummyStrings : [String]
  var pickerStrings : [String]
  var tmpXmlStrings : [String]

  @IBAction func dataSourceChangePressed(_ sender: UISegmentedControl)
  @IBAction func testOperationQueueButtonPressed(_ sender: UIButton)

  func getXMLData() -> [String]
  func getJSONData() -> [String]
}
```

2. **Dummy Picker-Daten**
   Im Picker sollen bei Auswahl von "Dummy" testweise die drei Test-Texte "One", "Two" und "Three" angezeigt werden. Verwenden sie dazu das Property `dummyStrings`, welches sie mit den drei Dummy-Texten füllen. Das Array `pickerStrings` soll immer ausschliesslich String-Objekte enthalten und zwar eben genau die aktuell in der PickerView darzustellenden Texte. Stellen sie also sicher, dass nach `viewDidLoad` das Array `pickerStrings` auf das Array `dummyStrings` referenziert. Setzen sie dazu nun den ViewController als delegate und dataSource von der PickerView. Am einfachsten geht das im Interface Builder, siehe Screenshot unten rechts. Damit nun die entsprechenden Daten auch angezeigt werden, müssen für die



PickerView die entsprechenden Delegate- und DataSource-Methoden implementiert werden. Implemtieren sie dann in der ViewController-Klasse die notwendigen delegate- und dataSource-Methoden. Daraufhin sollten in der PickerView die Dummy-Texte wie gewünscht angezeigt werden, wie im Screenshot rechts.



3. **XML-Daten im Internet holen, parsen und darstellen**
   Wenn in der SegmentedControl "XML" angeklickt wird (IBAction-Methode `dataSourceChangePressed`), sollen aktuelle Daten in einem einfachen XML-Format von https://wherever.ch/hslu/iPhoneAdressData.xml geholt und geparst werden. Die entsprechenden String-Objekte sollen am Schluss im Property `pickerStrings` abgelegt sein. Implementieren sie dazu in der ViewController-Klasse die Methode `getXMLData` und die Methode `parser:didStartElement:namespaceURI:qualifiedName:attributes:` vom Protokoll `XMLParserDelegate`. Verwenden sie im weiteren das Property `tmpXmlArray` um darin nach und nach die geparsten Daten abzulegen und dann am Schluss in einem Array zurückliefern zu können.

4. **JSON-Daten im Internet holen, parsen und darstellen**
   Analog zu den XML-Daten sollen nun entsprechend beim Anwählen von "JSON" auf der SegmentedControl von https://wherever.ch/hslu/iPhoneAdressData.json Daten in einem einfachen JSON-Format geholt und dargestellt werden. Verwenden sie dazu in der Methode `getJSONData` u.a. die Klasse `JSONSerialization` um die JSON-Textdatei zu parsen und in JSON-Objekte umzuwandeln. Hinweis: Die unter obiger Adresse abgelegte JSON-Datei besteht Top-Level aus einem Array, welches JSON-Objekte enthält. Extrahieren sie also entsprechend die gewünschten Daten aus dieser JSON-Datei.

5. **Operation Queues: BlockOperations und Abhängigkeiten**
   Die Methode `testOperationQueueButtonPressed` soll durch Drücken des entsprechenden Knopfes ausgelöst werden. Erzeugen sie in dieser Methode 3 Instanzen von `BlockOperation`. Bennen sie diese `blockOp1`, `blockOp2` und `blockOp3`. Diese drei Operationen tun nichts anderes, als einem in dieser Methode deklarierten, instanziierten und veränderbaren Array `orderArray` einen String mit ihrer Nummer einfügen. Der gesamte Closure-Inhalt von `blockOp1` sieht also wie folgt aus (und analog für die anderen beiden Operations):
   `{ orderArray.append("1") }`
   Analog besteht der Block-Inhalt in Swift aus folgender einfachen Closure:
   `{orderArray.append("1")}`
   Im weiteren sollen sie die drei Operationen wie folgt voneinander abhängig sein: `blockOp1` hängt von `blockOp2` und `blockOp3` ab, und `blockOp2` hängt von `blockOp3` ab. Setzen sie diese Abhängigkeiten mit Hilfe der `NSOperation`-Methode `addDependency`. Erzeugen sie dann eine Instanz von `OperationQueue` und fügen sie dieser Queue die drei Operationen in einem Array verpackt hinzu. Geben sie anschliessend die Reihenfolge der Ausführung der Operationen, auf welche sie durch die String-Objekte in `orderArray` schliessen können, in einer AlertView aus. Das sollte dann aussehen wie im Screenshot rechts.