CSC 491 / 391 Mobile Application Development for iOS II

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

Outline

- Asynchronous calls
- Singleton design pattern
  - Access levels
- Lazy loading
- Parsing JSON data
- CocoaPods

2

Asynchronous & Lazy Loading

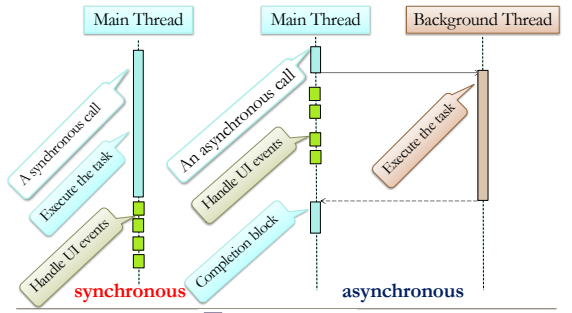
3

Synchronous Calls vs. Asynchronous Calls

- Synchronous calls
  - Blocking calls. Wait for tasks to complete.
  - Execution on the same or a different thread.
  - Return upon completion.
- Asynchronous calls
  - Non-blocking calls. Return immediately.
  - Execution on a different tread.
  - Call back upon completion
    - Commonly in Swift, *completion blocks*

4

Synchronous vs. Asynchronous



6

An ITunes RSS Reader with a Single Entry

7

An ITunes RSS Reader – the First Iteration

- Access iTune data, e.g., top paid apps
- (Asynchronous) HTTP request
  - Data returned in XML or JSON
- Parse JSON
- Handle errors

DePaul University 7

8

Failable Initializers

- An initializer that may fail
  - Declared as `init?(…)`
  - Return `nil` when initialization fails
  - The result is an optional value
- Example:

```
class Car {
    let model: String
    init?(_ model: String) {
        if model.isEmpty {
            return nil
        }
        self.model = model
    }
}

let myCar = Car("Tesla 3S")
myCar?.model
let anotherCar = Car("")
anotherCar?.model
```

The type is `Car?`

DePaul University 8

11

Handle HTTP Requests

```
let feed = "http://ax.itunes.apple.com/…/limit=1/json"
…
guard let feedURL = URL(string: feed)
    else { … return }
let request = URLRequest(url: feedURL)
let session = URLSession.shared
session.dataTask(with: request) {
       data, response, error in
    Process the data received from the request
}.resume()
```

The request URL

A failable initializer

The shared instance, singleton

An asynchronous call

A completion handler of the asynchronous call, as a closure

DePaul University 9

14

URL Session

- Load and manage the contents of a URL
  - Download or upload data
  - Manage cookies, caching, credentials, and authentication
- Natively supported URL schemes
  - File Transfer Protocol – ftp://
  - Hypertext Transfer Protocol – http:// https://
  - Local file URLs – file:///
  - Data URLs – data://

DePaul University 10

15

Using an URL Session

- Create one or more *sessions*
  - A shared session, singleton, handle basic requests
  - Other sessions, customizable and configurable
    - *Default, Ephemeral,* and *Background* sessions
- Create tasks in a session
  - *Data* tasks, short, interactive, send/receive data
  - *Upload* and *Download* tasks, background processing
- Each task starts in a *suspended* state.
  - Call `resume()` to begin receiving/sending data

DePaul University 11

16

JSON

- **JSON** (JavaScript Object Notation)
- A lightweight data-interchange format
- Widely used and supported
- Compared to XML
  - More compact, less verbose, and more readable
  - Faster to parse
  - Limited data types

DePaul University 12

17

Examples of JSON Data

An Array

```
[
   "Hello world!",
   2021,
   true
]
```

A Dictionary

```
{
  "name": "Caffè Macs",
  "coordinates": {
    "lat": 37.330576,
    "lng": -122.029739
  },
  "menu": [
     "breakfast",
     "lunch",
     "dinner"
  ]
}
```

A Dictionary in a Dictionary

An Array in a Dictionary

DePaul University 13

18

JSON Serialization

- Class `JSONSerialization`
  - Converts JSON data into Swift data types:
    - `Dictionary`, `Array`, `String`, `Number`, and `Bool`, etc.
- Extracting values from JSON
  - Dictionary: `[String:Any]`
  - Array: `[Any]`
  - Values: `String` `Number` `Bool`
- JSON data can be non-uniform
  - Optional downcasting is often necessary

DePaul University 14

19

Non-Specific Types

- Two special types to represent values and objects of non-specific types

  `Any`
  - an instance of any type, including class, struct, enum, and closure types.

  `AnyObject`
  - an instance of any class or closure type.
- Not classes, but protocols
- Use specific types whenever possible

DePaul University 15

20

Process JSON Data

- Deserialize JSON data

```
let json = try? JSONSerialization.jsonObject(
                                with: data, options: [])
```

- When the root is a dictionary

```
if let dictionary = json as? [String: Any] { … }
```

- When the root is an array

```
if let array = json as? [Any] { … }
```

- Extract the data descending the tree, one level at a time
- May encounter various types of anomalies in data
  - missing keys
  - invalid values

```
enum SerializationError: Error {
    case missing(String)
    case invalid(String, Any)
}
```

DePaul University 16

21

Receiving Data from URL Session

```
session.dataTask(with: request) { data, response, error in
    guard error == nil else {
        print(error!.localizedDescription); return
    }
    guard let data = data else { return }
    do {
        …
    } catch SerializationError.missing(let msg) {
        print("Missing \(msg)")
    } catch SerializationError.invalid(let msg, let data) {
        print("Invalid \(msg): \(data)")
    } catch let error as NSError {
        print(error.localizedDescription)
    }
}.resume()
```

An NSError

Parse JSON data

DePaul University 17

22

JSON Data from iTunes

```
{ "feed": {
    "author": { ... },
    "name": { ... },
    "entry": {
        "im:artist": { "attributes": { "href": "https://…/…" },
                       "label": "Mojang"},
        "im:contentType": { … },
        "im:image": [
            { "attributes": { "height": 53 }, "label": "http://….png" },
            { "attributes": { "height": 75 }, "label": "http://….png" },
            { "attributes": { "height": 100 }, "label": "http://….png" }
        ],
        "im:name": { "label": "Minecraft: Pocket Edition" }
    }
  },
  ...
}
```

DePaul University 18

23

Parsing JSON Data

```
if let json = try JSONSerialization.jsonObject(with: data!, options: [])
                    as? [String:Any] {
    guard let feed = json["feed"] as? [String:Any] else {
        throw SerializationError.missing("feed")
    }
    guard let entry = feed["entry"] as? [String:Any] else {
        throw SerializationError.missing("entry")
    }
    guard let nameEntry = entry["im:name"] as? [String:Any] else {
        throw SerializationError.missing("im:name")
    }
    guard let artistEntry = entry["im:artist"] as? [String:Any] else {
        throw SerializationError.missing("im:artist")
    }
    guard let imageArray = entry["im:image"] as? [Any] else {
        throw SerializationError.missing("im:image")
    }
    …
```

{ "feed": … }

{ "entry": … }

{
"im:name": …
"im:artist": …
"im:image": …
}

DEPAUL UNIVERSITY 19

24

Parsing & Displaying Data

```
if let json = try JSONSerialization.jsonObject() as? [String:Any] {
    guard let feed = …
    guard let entry = …
    guard let nameEntry = …
    guard let artistEntry = …
    guard let imageArray = …
    if let title = nameEntry["label"] as? String,
       let artist = artistEntry["label"] as? String {
        DispatchQueue.main.async {
            self.titleLabel.text = title
            self.artistLabel.text = artist
        }
        if let imageEntry = imageArray.last as? [String:Any],
           let imageURL = imageEntry["label"] as? String,
           let url = URL(string:imageURL) {
            self.loadImage(from: url, in: self.imageView)
        }
    }
}
```

{ "label": "…" }

{ "label": "…" }

[{…} {…} …]

{ "label": "…" }

DEPAUL UNIVERSITY 20

25

Loading Image

- A second asynchronous call

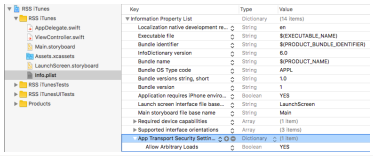
```
func loadImage(from URL: Foundation.URL,
               in imageView: UIImageView) {
    let request = URLRequest(url: URL)
    URLSession.shared.dataTask(with: request) {
            data, response, error in
        if let imageData = data {
            let image = UIImage(data: imageData)
            DispatchQueue.main.async {
                imageView.image = image
            }
        }
    }.resume()
}
```

DEPAUL UNIVERSITY 21

26

App Transport Security (ATS)

- Require secure network connections over HTTPS
  - Improves user security and privacy
- Add a new entry in your app's `Info.plist` file
  - `Allow Arbitrary Loads: YES`



DEPAUL UNIVERSITY 22

27

Design Pattern – Singleton

28

The Singleton Pattern

- One of the most common design patterns
- To ensure a unique instance, i.e., shared instance, for the entire application.
  - No way to create a second instance, intentionally or accidentally
  - Accessing the shared instance is thread-safe.
- Lazy initialization
  - Initialized when the instance is first accessed

DEPAUL UNIVERSITY 24

29

The Singleton Pattern

```
class MySingleton {

    static let sharedInstance = MySingleton()

    private init() {}

    var data: String = "foo"
    // other members …
}

let myinstance = MySingleton.sharedInstance
print(myinstance.data)
```

The shared instance

Initialization of static and global variables are *lazy* and *atomic*

The initializer is not accessible outside the class

The instance initialized here

30

Swift Access Control

- Access levels are defined in terms of the various levels of units of your code,
  - *Modules:* a single unit of distribution
    - Typically consists of multiple source files.
    - A simple project usually consists of a single model.
  - *Source files:* a file containing Swift source code.
    - May contain definitions of multiple classes, functions, etc.
  - *Scopes:* a section of source code within a single source file delimited by a matching pair of { and }.
    - E.g., the definition of a single class.

31

Access Levels

- `open` – accessible anywhere
  - Only applicable to classes and class members
  - Subclassing and overriding allowed anywhere.
- `public` – accessible anywhere
  - Subclassing allowed only within its own module
- `internal` – accessible only within its own module
  - The default access level
- `fileprivate` – accessible only within its own file
- `private` – accessible only within the enclosing scope of its definition, and *extensions in the same file*

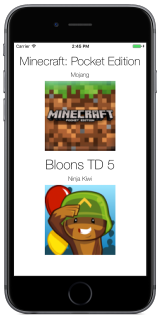
32

An ITunes RSS Reader with Multiple Entries

33

The Second Iteration – Loading Multiple Entries

- Two entries
- Loading an array from JSON data



34

The UI and the Outlets

36

Parsing JSON Data

```
let feed = "http://ax.itunes.apple.com/…/limit=2/json"
…
if let json = try JSONSerialization.jsonObject() as? [String:Any] {
  guard let feed = json["feed"] as? [String:Any] else { throw … }
  guard let entries = feed["entry"] as? [Any] else {throw … }
  for (i, e) in entries.enumerated() {
    if let entry = e as? [String:Any] {
      guard let nameEntry = entry["im:name"] as? [String:Any] else {…}
      guard let artistEntry = entry["im:artist"] as? [String:Any] else {…}
      guard let imageArray = entry["im:image"] as? [Any] else {…}
      if let title = nameEntry["label"] as? String,
         let artist = artistEntry["label"] as? String {
        DispatchQueue.main.async {
            self.titleLabels[i].text = title
            self.artistLabels[i].text = artist
        }
        …
```

{ "feed": … }

{ "entry": … }

[{…} {…}]

37

Parsing JSON Data

```
if let json = try JSONSerialization.jsonObject() as? [String:Any] {
  guard let feed = …
  guard let entry = …
  for (i, e) in entries.enumerated() {
    if let entry = e as? [String:Any] {
      guard let nameEntry = entry["im:name"] as? [String:Any] else {…}
      guard let artistEntry = entry["im:artist"] as? [String:Any] else {…}
      guard let imageArray = entry["im:image"] as? [Any] else {…}
      if let title = nameEntry["label"] as? String,
         let artist = artistEntry["label"] as? String {
        …
        if let imageEntry = imageArray.last as? [String:Any],
           let imageURL = imageEntry["label"] as? String,
           let url = URL(string:imageURL) {
            self.loadImage(from: url, in: self.imageViews[i])
        }
      }
    }
  }
}
```

38

An ITunes RSS Reader with a Lazy Loading Table

39

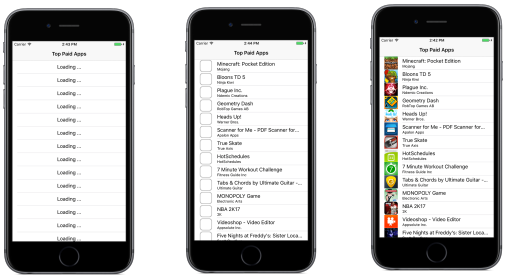
Lazy Loading

- Load resource only when it is needed, e.g., becomes visible
- Use asynchronous calls to perform time-consuming loading tasks
  - Avoid blocking the main thread
  - Keep UI responsive
- Improve perceived performance, and user experience (UX)
  - Quick, meaningful feedback

40

The Third Iteration – A Lazy Loading Table View

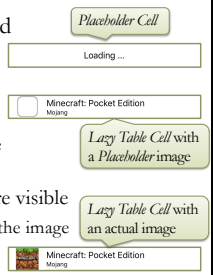
- A table view with three stages of loading



41

A Lazy Loading Table View

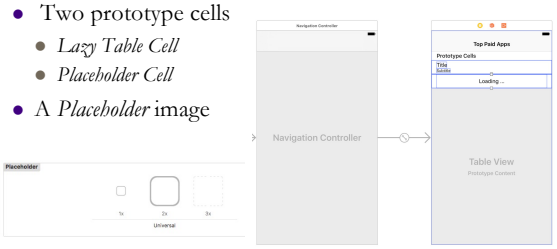
- A table view displaying iTunes data
- Lazy loading contents, on-demand
  - Initial loading
    - placeholder cells, no text, no image
  - Text only
    - when the text data is available, image displayed using a placeholder
  - Images are loaded when the cells are visible
    - replace the image placeholder when the image is available



42

The Table View Design

- A navigation controller and a table view controller
- Two prototype cells
  - *Lazy Table Cell*
  - *Placeholder Cell*
- A *Placeholder* image



DePaul University 37

43

The Table View Controller

```
let feed = "http://ax.itunes.apple.com/…/limit=75/json"

class TableViewController: UITableViewController {
    class Record {
        var title: String = ""
        var artist: String = ""
        var imageURL: URL?
        var icon: UIImage?
    }

    var dataAvailable = false
    var records: [Record] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        loadData()
    }
    …
}
```

A class to stored the data for each entry

Optional type for lazy loading

No data is available initially

Start loading data when view loading is complete

DePaul University 38

44

The Table View Controller

- The number of rows of the table view

```
class TableViewController: UITableViewController {
    …
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView,
                            numberOfRowsInSection section: Int) -> Int {
        // if there's no data yet, return enough rows to fill the screen
        return dataAvailable ? records.count : 15
    }
    …
}
```

DePaul University 39

45

The Initial View – No Data

- Before data loading is complete
  - Showing the *Placeholder Cells* for every row

```
class TableViewController: UITableViewController {
  override func tableView(_ tableView: UITableView,
                          cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    if (dataAvailable) {
      …
    } else {
      let cell = tableView.dequeueReusableCell(
                    withIdentifier: "PlaceholderCell", for: indexPath)
      return cell
    }

  }
  …
}
```

DePaul University 40

46

Loading Data without Images

```
func loadData() {
  guard let feedURL = URL(string: feed) else { return }
  let request = URLRequest(url: feedURL)
  let session = URLSession.shared
  session.dataTask(with: request) { data, response, error in
    guard error == nil else { … }
    guard let data = data else { return }
    do {
      if let json = try JSONSerialization.jsonObject(…) as? [String:Any] {
        guard let feed = json["feed"] as? [String:Any] else {…}
        guard let entries = feed["entry"] as? [Any] else {…}
        for e in entries {
          if let entry = e as? [String:Any] {
            Process the data for a single entry (the next slide)
          }
        }
    } catch … { … }
  }.resume()
}
```

DePaul University 41

48

Loading Data without Images

```
for e in entries {
  if let entry = e as? [String:Any] {
    guard let nameEntry = entry["im:name"] as? [String:Any] else {…}
    guard let artistEntry = entry["im:artist"] as? [String:Any] else {…}
    guard let imageArray = entry["im:image"] as? [Any] else {…}
    if let title = nameEntry["label"] as? String,
       let artist = artistEntry["label"] as? String {
      let record = Record()
      record.title = title
      record.artist = artist
      if let imageEntry = imageArray.last as? [String:Any],
         let imageURL = imageEntry["label"] as? String {
          record.imageURL = URL(string:imageURL)
      }
      self.records.append(record)
    }
  }
}
self.dataAvailable = true
DispatchQueue.main.async{
   self.tableView.reloadData()
}
```

Only set the image URL, not the image

Data is available. Table view is reloaded with the data.

DePaul University 42

49

The Table View with Data (Without Image)

- After data loading is complete

```
override func tableView(_ tableView: UITableView,
                cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    if (dataAvailable) {
      let cell = tableView.dequeueReusableCell(
                  withIdentifier: "LazyTableCell", for: indexPath)
      let record = records[indexPath.row]
      cell.textLabel?.text = record.title
      cell.detailTextLabel?.text = record.artist
      if let icon = record.icon {
          cell.imageView?.image = icon
      } else {
          cell.imageView?.image = UIImage(named: "Placeholder")
          loadImage(for: record, in: cell.imageView)
      }
      return cell
    } else { … }
}
```

No image loaded yet. Use Placeholder

50

Loading Images and Updating

```
func loadImage(for record: Record, in imageView: UIImageView?) {
    let request = URLRequest(url: record.imageURL!)
    URLSession.shared.dataTask(with: request) {
            data, response, error in
        if let imageData = data {
            let image = UIImage(data: imageData)
            record.icon = image
            if let imageView = imageView {
                DispatchQueue.main.async {
                    imageView.image = image
                }
            }
        }
    }.resume()
}
```

Table view will be updated when image loading is done.

52

Handling Errors in Data

- Need to catch errors, recover and limit the damage

```
func loadData() {
  …
  session.dataTask(with: request) { data, response, error in
    guard error == nil else { … }
    guard let data = data else { return }
    do {
      if let json = try JSONSerialization.jsonObject(…) {
        …
      }
    } catch … { … }
  }.resume()
}
```

What happens when an error is caught?

54

Handling Errors in Data

```
func loadData() {
  …
  session.dataTask(with: request) { data, response, error in
    …
    do {
      if let json = try JSONSerialization.jsonObject(…) as? [String:Any] {
        guard let feed = json["feed"] as? [String:Any] else {…}
        guard let entries = feed["entry"] as? [Any] else {…}
        for e in entries {
          do {
            if let entry = e as? [String:Any] {
              …
            }
          } catch … { … }
        }
    } catch … { … }
  }.resume()
}
```

Limit the errors in a single entry. Skip to the next entry

55

Low Memory Warning

- Dispose of any resources that can be recreated

```
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    for r in records {
        r.icon = nil
    }
}
```

Dispose all the images and retain all the records.

- A more drastic alternative

```
override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    records.removeAll(keepingCapacity: false)
    dataAvailable = false
}
```

Dispose all the data.

56

Codable and JSON Parsing

57

Encoding & Decoding Protocols

- The Swift standard library defines a standardized approach to data encoding and decoding.
- The *Codable* protocol

```
typealias Codable = Decodable & Encodable
```

- The *Encodable* and *Decodable* protocols

```
protocol Encodable
    func encode(to encoder: Encoder) throws
}
protocol Decodable {
    init(from decoder: Decoder) throws
}
```

DePaul University 58

58

Encode and Decode Automatically

- Declare conformance to *Codable*
- Use properties that are codable
  - Most standard types are codable
    - String
    - Int
    - Double
    - Date
    - URL
    - Array
    - Dictionary
    - etc.

```
struct Student : Codable {
    var name: String
    var id: Int
    var major: Program
}

struct Program : Codable {
    var name: String
}
```

DePaul University 59

59

Encode in JSON

```
let compSci = Program(name: "Computer Science");
let john = Student(name: "John Appleseed", id: 7,
                   major: compSci)

let jsonEncoder = JSONEncoder()
let jsonData = try jsonEncoder.encode(john)

let jsonString = String(data: jsonData, encoding: .utf8)
print(jsonString!)
```

Output:

```
{ "name" : "John Appleseed",
  "id" : 7,
  "major" : { "name" : "Computer Science" } }
```

DePaul University 60

60

Decode from JSON

```
…
let jsonDecoder = JSONDecoder()
let student2 = try jsonDecoder.decode(Student.self,
                                      from: jsonData)
print(student2)
```

Output:

```
Student(name: "John Appleseed", id: 7,
        major: Program(name: "Computer Science"))
```

DePaul University 61

61

Customize Coding Keys

- Declare a special nested enumeration named *Coding Keys*

```
struct Student : Codable {
    var name: String
    var id: Int
    var major: Program

    enum CodingKeys: String, CodingKey {
        case id = "studentId"
        case name
        case major
    }
}
```

```
{ "name" : "John Appleseed",
  "studentId" : 7,
  "major" : { "name" : "Computer Science"} }
```

DePaul University 62

62

Encode and Decode Manually

- Automatic encoding and decoding assumes the structures of Swift type and JSON data coincide.
- If the structures of the Swift type and JSON data are different, you need to provide custom implementation of encode and decode logic.

DePaul University 63

63

Encode Manually

```
struct Student : Encodable {
    var name: String
    var id: Int
    var major: Program
    enum CodingKeys: String, CodingKey {
        case id = "studentId"
        case name
        case major
    }
    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(name, forKey: .name)
        try container.encode(id, forKey: .id)
        try container.encode(major.name, forKey: .major)
    }
}
```

```
{ "name" : "John Appleseed",
  "studentId" : 7,
  "major": "Computer Science" }
```

DePaul University 64

64

Decode Manually

```
extension Student: Decodable {
    init(from decoder: Decoder) throws {
        let values = try decoder.container(keyedBy: CodingKeys.self)
        name = try values.decode(String.self, forKey: .name)
        id = try values.decode(Int.self, forKey: .id)
        let majorName = try values.decode(String.self,
                                          forKey: .major)
        major = Program(name: majorName)
    }
}
```

```
{ "name" : "John Appleseed",
  "studentId" : 7,
  "major": "Computer Science" }
```

DePaul University 65

65

Encode More Data Types

```
struct Student : Codable {
    var name: String
    var id: Int
    var major: Program
    var minor: Program?
    var gpa: Double
    var category: Category
}
enum Category : Int, Codable {
    case Freshman, Sophomore, Junior, Senior
    case Graduate
}
let compSci = Program(name: "Computer Science");
let art = Program(name: "Art History");
let john = Student(name: "John Appleseed", id: 7,
                  major: compSci, minor: art,
                  gpa: 3.5, category: .Sophomore)
```

```
{   "name": "John Appleseed",
    "category": 1,
    "id": 7,
    "gpa": 3.5,
    "major": {
        "name": "Computer Science"
    },
    "minor": {
        "name": "Art History"
    }
}
```

DePaul University 66

66

Encode Collection Types

```
struct Student : Codable {
    var name: String
    var id: Int
    var major: Program
    var minor: Program?
    var gpa: Double
    var category: Category
    var courses: [Course]
    var grades: [String:Grade]
}
```

```
struct Course: Codable {
    var code: String
    var title: String
}
enum Grade: String, Codable {
    case A, B, C, D, F
}
```

DePaul University 67

67

Encode Collection Types

```
let cs101  = Course(code: "CS101",
                    title: "Intro to Swift")
let art201 = Course(code: "ART201",
                    title: "Renaissance Masterpieces")
let compSci = Program(name: "Computer Science");
let art = Program(name: "Art History");
let john = Student(name: "John Appleseed", id: 7,
                  major: compSci, minor: art,
                  gpa: 3.5, category: .Sophomore,
                  courses: [ cs101, art201],
                  grades: [ "CS101": .A, "ART201": .B])
```

DePaul University 68

68

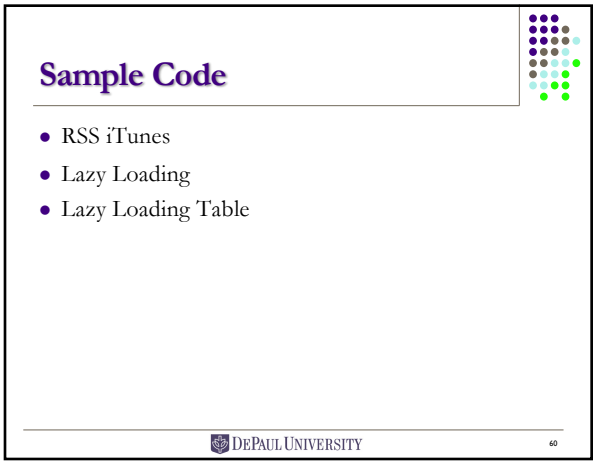
Encode Collection Types

- The JSON data

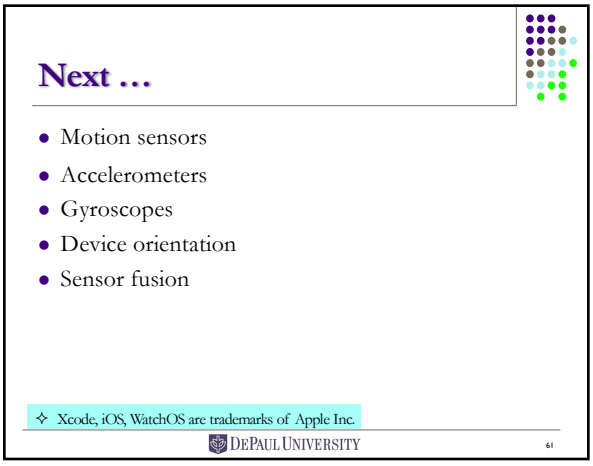
```
{
    "name": "John Appleseed",
    "category":1,
    "id":7,
    "gpa":3.5,
    "major": { "name":"Computer Science" },
    "minor": { "name":"Art History" },
    "grades": {
        "CS101": "A",
        "ART201": "B"
    },
    "courses":
    [
      { "title": "Intro to Swift",
        "code": "CS101" },
      { "title": "Renaissance Masterpieces",
        "code":"ART201"}
    ],
}
```

DePaul University 69

69


Sample Code
RSS iTunes
Lazy Loading
Lazy Loading Table
DePaul University
60

70


Next …
Motion sensors
Accelerometers
Gyroscopes
Device orientation
Sensor fusion
Xcode, iOS, WatchOS are trademarks of Apple Inc.
DePaul University
61

71