Intermediate Realm on iOS

Hands-On Challenges

Intermediate Realm on iOS

Marin Todorov

Table of Contents: Overview

Table of Contents: Extended

6 Challenges: Migrations Part 2

By Marin Todorov

Challenge A: Add more statuses

Let’s add some more statuses to the exams.

First you will add a new status called “just added” to each newly created `Exam`. Open **SubjectsViewController.swift** and find the place in the code where you create a new `Exam` instance.

Right after you get a reference to your Realm add:

```
let justAddedText = "just added"

if let justAdded = realm.objects(Status.self).filter("status == %@",
justAddedText).first {
  exam.statuses.append(justAdded)
} else {
  exam.statuses.append(Status(justAddedText))
}
```

First you define the text of the new status and then you check if a Status like it already exists. If so – you go ahead and just add the found object as a status of the newly created `Exam`; if not – you create a new `Status` with the text in question and add it directly to the `Exam`.

When you run the app now – each exam you add will have this new status too.

You will define one more status for your exams. Pretend that with the release of Exams v.1.2 you noticed that your v1.0 to v.1.2 data migration broke some of the functionality by adding the text “(multiple choice)” directly to the name of the exams.

Now if you go to the list of subjects you will see a “Math 101” subject, because the exam name is actually “Math 101(multiple choice)” and your filter does not work correctly anymore.

Open **Exams.swift** and scroll to your migration code then find the offensive code:

```
if let oldObject = oldObject,
    let multi = oldObject["multipleChoice"] as? Bool, multi {
    newObject["name"] = "\(newObject["name"]!)(multiple choice)"
}
```

It certainly looked as a great idea at the time but as the app evolves you need to evolve your migration code as well :]

Luckily in Exams v.2.0 you have separate `Status` class, which you can use to denote that certain exams are multiple-choice and others aren't.

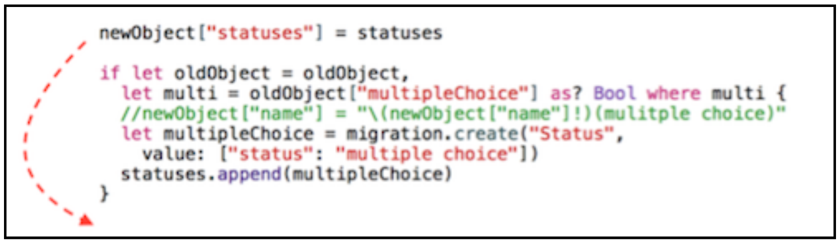
Remove the line that sets `newObject[“name”]` and insert in its place:

```
let multipleChoice = migration.create("Status", value: ["status":
"multiple choice"])
statuses.append(multipleChoice)
```

You create a new `MigrationObject` with one key called “status” and its value being “multiple choice”. You append this object to the list of statuses and you're off to the races – all your problems are fixed for ever!

Pay attention because it's up to you to provide the correct class name and property names to the `create(_:, value:)` method.

Have a look at the code again - in case you aren't setting the "statuses" key as the very last thing in that block, move it after the code creating the multiple choice status, like so:



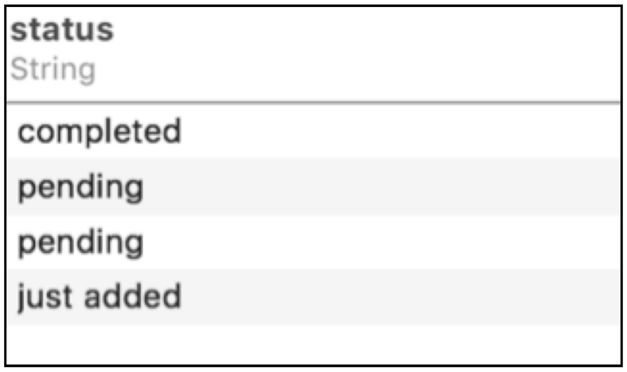
```
newObject["statuses"] = statuses

if let oldObject = oldObject,
  let multi = oldObject["multipleChoice"] as? Bool where multi {
  //newObject["name"] = "\(newObject["name"]!)(mulitple choice)"
  let multipleChoice = migration.create("Status",
    value: ["status": "multiple choice"])
  statuses.append(multipleChoice)
}
```

If you compare the two situation where you add “just added” and “multiple choice” status objects you will notice one core difference.

When you add a `Status` object normally you can query the Realm for an existing copy of the status before adding a new one. In your migration block you can't query the `Status` objects that you're adding during the migration. This results in a new `Status` object being added for each multiple choice `Exam`.

If you delete the app and run the project you will see duplicate status objects in your app's Realm file:



status String
completed
pending
pending
just added

It seems I spoke too soon when I said this challenge's code solves all your problems for ever :] Move on to the next challenge to explore a possible solution to the multiple choice status problem.

Challenge B: Keeping track of status objects added during migration

Issues like the repeated "multiple choice" statuses are non-trivial and highly dependent on the logic of your app so you need to think about possible solution on a case by case basis.

In this challenge you are going to fix the issue in the concrete Exams app project but keep in mind the code is suitable for the specific case.

During migration you create `Status` objects by invoking the `create(_:, value:)` method on the active migration. The result of that method is of type `MigrationObject`, which is just a data placeholder that the migration uses to create the final objects in your Realm when the migration process is finished.

To keep track of the objects you create during migration you can store them in an array or a dictionary. For this challenge you are going to use this approach since the objects are relatively small in size and there's no danger of running out of memory during migration.

Open **Exams.swift** and add a new static property to the class:

```
fileprivate static var migratedStatuses: [String: MigrationObject]?
```

You can add the property to the class or create it inside the migrate method – in this challenge you are going to do the former since the code is a tad easier to read that way.

You will store all statuses you create during the migration in `migratedStatuses` and re-use objects from that dictionary if you have any repeating status messages.

Next create a method that will either create a new `Status` or re-use existing one:

```
fileprivate static func addOrReuseStatus(_ migration: Migration, text:
String) -> MigrationObject {

}
```

This method takes in a `Migration` instance and a status text and returns a `MigrationObject` – either a new one or one that the method found in `migratedStatuses`.

Now you need to simply check if an object has been already created for the given status text:

```
if let existingStatus = migratedStatuses?[text] {
  return existingStatus
} else {

}
```

In case this is a new status you will need to create a new `Status` object (just as you did earlier) and store it in `migratedStatuses`. Insert in the `else` branch:

```
let status = migration.create("Status", value: ["status": text])
migratedStatuses?[text] = status
return status
```

So far so good! Next you have to adjust your existing code to use the new method.

Insert at the top of `migrate(_:, fileSchemaVersion:)`:

```
migratedStatuses = [:]
```

This will initialize your cache dictionary and “enable” the usage of `addOrReuseStatus`.

Now scroll down and at the very bottom of that the method insert:

```
migratedStatuses = nil
```

Mind the position of the two lines - initializing the cache dictionary should happen before you call `migration.enumerate(_)` and setting it to `nil` should happen right after the call to `migration.enumerate(_)`.

Now you can change the code around re-using existing status objects.

Find this line:

```
let completeness = migration.create("Status", value: ["status":
statusText])
```

And replace it with:

```
let completeness = addOrReuseStatus(migration, text: statusText)
```

Instead of creating a new object each time you call `addOrReuseStatus(_:, text:)`.

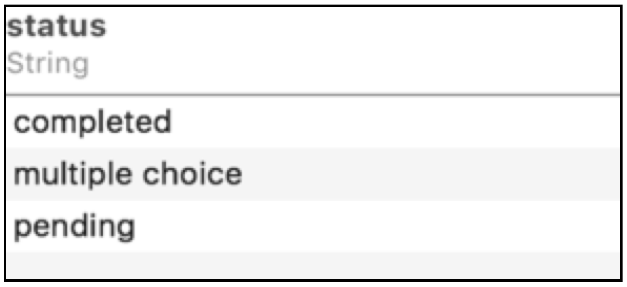
Next find the other line where you create statuses during migration:

```
let multipleChoice = migration.create("Status", value: ["status":
"multiple choice"])
```

And replace it with:

```
let multipleChoice = addOrReuseStatus(migration, text: "multiple choice")
```

With these last change the migration should work just fine – if you delete the app from the iPhone Simulator and run it again this time it will not create any duplicate status objects:



status String
completed
multiple choice
pending