

Итоговая часть по CoreData

Для упрощения работы создаю синглтон.

Одиночка (англ. **Singleton**) — порождающий шаблон проектирования, гарантирующий, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

```
import CoreData
import Foundation

class CoreDataManager {

    // Singleton
    static let instance = CoreDataManager()

    private init() {}

    // MARK: - Core Data stack

    lazy var persistentContainer: NSPersistentContainer = {
        /*
         The persistent container for the application. This implementation
         creates and returns a container, having loaded the store for the
         application to it. This property is optional since there are legitimate
         error conditions that could cause the creation of the store to fail.
        */
        let container = NSPersistentContainer(name: "qwerty")
        container.loadPersistentStores(completionHandler: { (storeDescription, error) in
            if let error = error as NSError? {
                // Replace this implementation with code to handle the error appropriately.
                // fatalError() causes the application to generate a crash log and terminate. You should not use this
                // function in a shipping application, although it may be useful during development.

                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        return container
    }()

    // MARK: - Core Data Saving support

    func saveContext () {
        let context = persistentContainer.viewContext
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                // Replace this implementation with code to handle the error appropriately.
                // fatalError() causes the application to generate a crash log and terminate. You should not use this
                // function in a shipping application, although it may be useful during development.
                let nerror = error as NSError
                fatalError("Unresolved error \(nerror), \(nerror.userInfo)")
            }
        }
    }
}
```

В тот же класс CoreDataManager кладу следующий метод **NSEntityDescription** — как можно догадаться из названия, это объект, который содержит описание нашей сущности. Все то, что мы нафантазировали с сущностью в Редакторе модели данных (атрибуты, взаимосвязи, правила удаления и прочее), содержится в этом объекте. Единственное, что мы будем делать с ним — получать его и передавать куда-то в качестве параметра, больше ничего

```
// Entity for Name

func entityForName(entityName: String) -> NSEntityDescription {
    return NSEntityDescription.entity(forEntityName: entityName, in: self.persistentContainer.viewContext)!
}
```

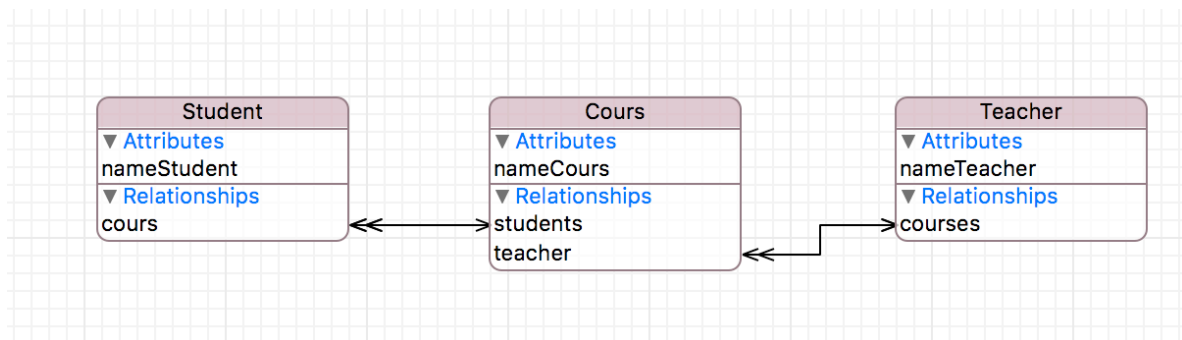
В самом классе сущности создаю новый объект согласно
NSEntityDescription

```
import Foundation
import CoreData

public class Student: NSManagedObject {

    convenience init() {
        self.init(entity: CoreDataManager.instance.entityForName(entityName: "Student"), insertInto: CoreDataManager.
            instance.persistentContainer.viewContext)
    }
}
```

Создаю модель



```
let managedObjectStud1 = Student()
managedObjectStud1.nameStudent = "Sasha"

let managedObjectStud2 = Student()
managedObjectStud2.nameStudent = "Ivan"

let managedObjectStud3 = Student()
managedObjectStud3.nameStudent = "Roman"


let managedObjectCours1 = Cours()
managedObjectCours1.nameCours = "Swift"

let managedObjectCours2 = Cours()
managedObjectCours2.nameCours = "ObjectiveC"



let managedObjectTeacher = Teacher()
managedObjectTeacher.nameTeacher = "Alexsej"


managedObjectStud1.cours = managedObjectCours1
managedObjectCours1.addToStudents(managedObjectStud2)

managedObjectStud3.cours = managedObjectCours2


managedObjectTeacher.addToCourses(managedObjectCours1)
managedObjectCours2.teacher = managedObjectTeacher


CoreDataManager.instance.saveContext()
```



Извлечение

Вернет массив согласно условий сортировки и предиката

```
// Извлечение записей

func resultRequest(entityName: String, sortKey: String!) -> [Any] {
    var results = [Any]()
    let fetchRequest = NSFetchRequest<NSFetchRequestResult>(entityName: entityName)
    let sortDescriptor = NSSortDescriptor(key: sortKey, ascending: true)
    fetchRequest.sortDescriptors = [sortDescriptor]
    let pred = NSPredicate(format: "nameStudent == %@", "Alex")
    fetchRequest.predicate = pred
    do {
        results = try CoreDataManager.instance.persistentContainer.viewContext.fetch(fetchRequest)
    } catch {
        print(error.localizedDescription)
    }
    return results
}
```

Любые изменения вносимые в объекты в этом массиве будут отслеживаться контекстом управляемых объектов (посредством отправки этому контексту сообщения save:)

```
resultRequest = CoreDataManager.instance.resultRequest(entityName: "Student", sortKey: "nameStudent")
(resultRequest[1] as! Student).nameStudent = "newText"
CoreDataManager.instance.saveContext()
```

NSFetchedResultsController

```
import UIKit
import CoreData

class CustomersTableViewController: UITableViewController {
    var fetchedResultsController: NSFetchedResultsController = {
        let fetchRequest = NSFetchRequest(entityName: "Customer")
        let sortDescriptor = NSSortDescriptor(key: "name", ascending: true)
        fetchRequest.sortDescriptors = [sortDescriptor]
        let fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest, managedObjectContext:
            CoreDataManager.instance.managedObjectContext, sectionNameKeyPath: nil, cacheName: nil)
        return fetchedResultsController
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        do {
            try fetchedResultsController.performFetch()
        } catch {
            print(error)
        }
    }

    // MARK: - Table View Data Source

    override func tableView(tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        if let sections = fetchedResultsController.sections {
            return sections[section].numberOfObjects
        } else {
            return 0
        }
    }

    override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
        let customer = fetchedResultsController.objectAtIndexPath(indexPath) as! Customer
        let cell = UITableViewCell()
        cell.textLabel?.text = customer.name
        return cell
    }
}
```

В разделе определения переменных мы создаем объект

fetchedResultsController с типом **NSFetchedResultsController**. Как видите, он создается на базе **NSFetchRequest** (я создал **NSFetchRequest** на основании сущности «Customer» и задал сортировку по имени Заказчика). Затем мы создаем сам **NSFetchedResultsController**, передав в его конструктор **NSFetchRequest** и нужный нам управляемый контекст, дополнительные параметры конструктора (`sectionNameKeyPath`, `cacheName`) мы здесь использовать не будем.

МОЖНО перенести для удобства в CoreDataMasager в синглтон

```
// Fetched Results Controller for Entity Name
func fetchedResultsController(entityName: String, keyForSort: String) -> NSFetchedResultsController
{
    let fetchRequest = NSFetchRequest(entityName: entityName)
    let sortDescriptor = NSSortDescriptor(key: keyForSort, ascending: true)
    fetchRequest.sortDescriptors = [sortDescriptor]
    let fetchedResultsController = NSFetchedResultsController(fetchRequest: fetchRequest, managedObjectContext: CoreDataManager.instance.managedObjectContext, sectionNameKeyPath: nil, cacheName: nil)
    return fetchedResultsController
}
```

И добавим следующую реализацию протокола.

```
1 // MARK: - Fetched Results Controller Delegate
2
3 func controllerWillChangeContent(controller: NSFetchedResultsController) {
4     tableView.beginUpdates()
5 }
6
7 func controller(controller: NSFetchedResultsController, didChangeObject anObject: AnyObject, atIndexPath indexPath:
8     NSIndexPath?, forChangeType type: NSFetchedResultsChangeType, newIndexPath: NSIndexPath?) {
9     switch type {
10     case .Insert:
11         if let indexPath = newIndexPath {
12             tableView.insertRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
13         }
14     case .Update:
15         if let indexPath = indexPath {
16             let customer = fetchedResultsController.objectAtIndexPath(indexPath) as! Customer
17             let cell = tableView.cellForRowAtIndexPath(indexPath)
18             cell?.textLabel?.text = customer.name
19         }
20     case .Move:
21         if let indexPath = indexPath {
22             tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
23         }
24         if let newIndexPath = newIndexPath {
25             tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Automatic)
26         }
27     case .Delete:
28         if let indexPath = indexPath {
29             tableView.deleteRowsAtIndexPaths([indexPath], withRowAnimation: .Automatic)
30         }
31     }
32 }
33
34 func controllerDidChangeContent(controller: NSFetchedResultsController) {
35     tableView.endUpdates()
36 }
```

- **Insert** (добавление) — вставляем новую строку по указанному индексу (строка добавится не просто в конец списка, а в свое место в списке в соответствии с заданной сортировкой)
- **Update** (обновление) — данные объекта изменились, получаем строку из нашего списка по указанному индексу и обновляем информацию о ней
- **Move** (перемещение) — порядок строк изменился (например, Заказчика переименовали и он теперь располагается в соответствии с

сортировкой в другом месте), удаляем строку оттуда, где она была и добавляем уже по новому индексу

- **Delete** (удаление) — удаляем строку по указанному индексу.

УДАЛЕНИЕ

```
override func tableView(tableView: UITableView, commitEditingStyle editingStyle: UITableViewCellEditingStyle, forRowAtIndexPath indexPath: NSIndexPath) {  
    if editingStyle == .Delete {  
        let managedObject = fetchedResultsController.objectAtIndexPath(indexPath) as! NSManagedObject  
  
        CoreDataManager.instance.managedObjectContext.deleteObject(managedObject)  
        CoreDataManager.instance.saveContext()  
    }  
}
```