NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

HOMEWORK #3

dedicated to Research Seminar
«Immersion in iOS Development»

MOSCOW 2023

Theme: Add a feature to write down wishes to grant later.
Objective: Hone table view skills, and learn to store simple information in user defaults.

Task description

You will continue working on the WishMaker app. If you skipped this task ask the assistant to give you the best solution yet. You will add the second wish-granting feature: the power to store wishes to grant later! This requires you to use table view. You will be developing the same WishMaking app in HW4 too, so try and write the code in a way that won't haunt you in the future.

Task requirements

- You must delete Storyboard and use Swift + UIKit to complete this task.
- **You must use one of the following architectures: MVC, MVVM, or VIPER.**
- To use code from this tutorial or the internet you must understand it first.

Grading

Grade	Task
1	When run WishMakerViewController shows a write down button.
2	Pressing the button shows a new WishStoringViewController.
3	WishStoringViewController has a table view.
4	The table has cells.
5	Cells are devided in two groups: AddWishCell and WrittenWishCell.
6	Wishes created via AddWishesCell are added to the list of written wishes.
7	Written wishes are saved in UserDefaults and are shown on reoppening the app.
8	Written wishes can be deleted.
9	Written wishes can be edited.
10	CoreData used to save wishes or share wishes through share.

Disclaimer

Overdue submissions are **punished** by a decrease in the grade equal to the number of days past the due date. This punishment is limited to **minus 5 points**. Please keep this in mind and plan your week accordingly.

Ask all the questions regarding this task or iOS development in general. Our goal is to enable you to become **the best iOS developer** possible. Teachers and assistants are here to **help**!

Submissions containing extraordinary solutions, exceptional code style, pattern usage, and entertaining features can lead to more points added. This is non-negotiable. Repeating or copying solutions that earned bonus points previously does not grant bonus points.

Starting this homework **your bad code** will **decrease your grade**. Remember to remove magic numbers and write clean code.

Tutorial

Continue working on your project from the previous homework. But when making commits write HW-3: in your commit messages. Try committing every point separately for bonus points and glorious success later in (real) life.

Point 1:

You already know how to add a button. Let us do it properly together one more time for the last time.

Create an **addWishButton** like this:

```
private let addWishButton: UIButton = UIButton(type: .system)
```

Providing the type for variables, fields, and properties improves the build time of the project. Doing this automatically is a good habit not only in Swift. We also can see the type system in the initialization of the button. This adds a pressing animation to the button, which allows us to do less work. You like to do less work, right?

We want this button to be below the sliders, so we will have to change the **configureSliders** method and add to new methods. The first method is **configureAddWishButton**. It will place the button on the bottom of the screen and set its height, color, text, etc. I'll be using the pin layout, if you want to use vanilla constraints or SnapKit you'll have to do it yourself.

```
92.private func configureAddWishButton() {
93.    view.addSubview(addWishButton)
94.    addWishButton.setHeight(Constants.buttonHeight)
95.    addWishButton.pinBottom(to: view, Constants.buttonBottom)
96.    addWishButton.pinHorizontal(to: view, Constants.buttonSide)
97.
98.    addWishButton.backgroundColor = .white
99.    addWishButton.setTitleColor(.systemPink, for: .normal)
100.   addWishButton.setTitle(Constants.buttonText, for: .normal)
101.
102.   addWishButton.layer.cornerRadius = Constants.buttonRadius
103.   addWishButton.addTarget(self, action: #selector(addWishButtonPressed), for: .touchUpInside)
104.}
```

You will get build errors if you copy this code and paste it into your project. This is intentional. Add button constants yourself. On line 93 we add our button as the **WishMakerViewController** screen subview. Without it lines 94 to 96 that are setting the button position on the screen will cause a runtime exception. We love runtime exceptions, but in real projects, not in the homework project. Lines 98 to 100 are setting the button's colors. Line 102 gives our button rounded corners. Line 103 sets a function that will be called once the user lifts their finger inside the button.

The **addWishButtonPressed** is next:

```
@objc
private func addWishButtonPressed() {
    // this will be done later!
}
```

Change the **configureSliders** method to attach the slider bottom to the button's top. Run the app and see what it does.

Hopefully, you got a runtime exception for not calling **addWishButtonPressed** before **configureSliders**. Remember not to do that in the future. Once your button is added, it should look something like this:

Image 11

Point 2:

We already have the **addWishButtonPressed** method that will be called on the button tap. All we need to do is open a new screen. This can be done in two ways. The first is to present the screen using the **present** method. The second option is to add a navigation controller to our app. We will do the second option in the next homework, so let us present it. Before we can do it, we need to create a new file with a **WishStoringViewController** class. Don't forget to make it final and inherit it from the UIViewController.

```
final class WishStoringViewController: UIViewController {
    override func viewDidLoad() {
        view.backgroundColor = .blue
    }
}
```

We can present our new screen like this:

```
present(WishStoringViewController(), animated: true)
```

What this will do is it will put the **WishStoringViewController** screen above our main screen. This can be seen using the debug view hierarchy. You should know what it is and how to do it by now. With this, we can close our screen by swiping it down and dragging it by its top. This is not the best way to do it, and designers usually ask you to add a line on top (a commonly recognized indicator of a screen that can be closed by being dragged to the bottom), a back button, or a close button. To close the presented screen in the code, you can use the **.dismiss()** function. It is different from the navigation view controller approach.

Point 3:

To add a table view, let us create a method **configureTable():**

```
private let table: UITableView = UITableView(frame: .zero)

private func configureTable() {
    view.addSubview(table)
    table.backgroundColor = .red
    table.dataSource = self
    table.separatorStyle = .none
    table.layer.cornerRadius = Constants.tableCornerRadius

    table.pin(to: view, Constants.tableOffset)
}
```

Remember, functions go after overridden methods, but fields go before overridden methods. This code should cause an error on the line where we try to assign our screen as the table's data source. This is because to provide the data, our **WishStoringViewController** screen must confirm to the **UITableViewDataSource** protocol:

```
// MARK: - UITableViewDataSource
extension WishStoringViewController: UITableViewDataSource { … }
```

This should go under the original **WishStoringViewController** class. A new error will be shown, indicating that two methods are missing. Press the fix button inside the error, and the missing functions will be added to our extension. All extensions that extend a protocol are better marked, which is a good code practice. The two new methods that were inserted miss content. To figure out what each of them does, press on the unknown function with a right-click button and "Jump to definition." Apple's code usually has comments describing the function's purpose. To check whether you gained the point return 0 and **UITableViewCell()**, run the app and see the red table view.

Point 4:

To add cells we need to do several things. Let us start with creating a wish array with one temporary wish:

```
private var wishArray: [String] = ["I wish to add cells to the table"]
```

This array will store our wishes for the app session. Closing the app will empty the entries added during the session. Wishes added through the code will appear on every app launch, which is not ideal. We will deal with this later. We have the array, let us connect the number of wishes to the number of cells in the table:

```
return wishArray.count
```

This is excellent, but we want a **WrittenWishCell** instead of the default one.

Let us create a custom cell:

```
final class WrittenWishCell: UITableViewCell {
    static let reuseId: String = "WrittenWishCell"

    private enum Constants {
        static let wrapColor: UIColor = .white
        static let wrapRadius: CGFloat = 16
        static let wrapOffsetV: CGFloat = 5
        static let wrapOffsetH: CGFloat = 10
        static let wishLabelOffset: CGFloat = 8
    }

    private let wishLabel: UILabel = UILabel()

    // MARK: - Lifecycle
    override init(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        configureUI()
    }

    @available(*, unavailable)
    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func configure(with wish: String) {
        wishLabel.text = wish
    }

    private func configureUI() {
        selectionStyle = .none
        backgroundColor = .clear

        let wrap: UIView = UIView()
        addSubview(wrap)

        wrap.backgroundColor = Constants.wrapColor
        wrap.layer.cornerRadius = Constants.wrapRadius
        wrap.pinVertical(to: self, Constants.wrapOffsetV)
        wrap.pinHorizontal(to: self, Constants.wrapOffsetH)

        wrap.addSubview(wishLabel)
        wishLabel.pin(to: wrap, Constants.wishLabelOffset)
    }
}
```

The cell lifecycle includes the initializer, but there is also a **prepareForReuse** method. It is important since instead of creating new cells after scrolling tables are optimized to use the same cell over and over again. This means we should be very careful with cell views. We might accidentally add the same wishLabel each time. That would negatively affect the performance of the table, but the worst part is that the text of the previous wishes would be seen behind the current wish text.

To avoid that happening we arrange views once the cell is initialized in a **configureUI** method. We also make cosmetic preparations. The first one is to remove the selection style. Otherwise tapped wishes would become gray for no reason. The background of the cell is made clear to uncover the table beneath the cell. We also configure the label contents using a non-private configure method. Instead of a wish those methods usually intake a model.

The next step is to create a wish cell instead of our UITableView temporary solution. This is not as easy as one can imagine. To do so we need to dequeue a reusable cell.

This can be done as follows:

```
func tableView(
    _ tableView: UITableView,
    cellForRowAt indexPath: IndexPath
) -> UITableViewCell {
    let cell = tableView.dequeueReusableCell(
        withIdentifier: WrittenWishCell.reuseId,
        for: indexPath
    )

    guard let wishCell = cell as? WrittenWishCell else { return cell }

    wishCell.configure(with: wishArray[indexPath.row])

    return wishCell
}
```

Here we use the dequeueReusableCell method of the table view to create our cell. The issue is that it is created downcasted to the UITableViewCell. This means we cannot call the configure with wish method to add the wish to our wishLabel. We cast it to the **WrittenWishCell** class by optionally converting it to a child type. Run this code. See what appears on the screen. Hopefully, you did not read this line before running the app, cause you will no this code will lead to a runtime exception. Every runtime exception you get at this stage is crucial, it teaches you what is worth paying attention to when developing an iOS app. In this case, before dequeuing the cell we need to register it. Add the following line to the **configureTable()** method:

```
table.register(WrittenWishCell.self, forCellReuseIdentifier: WrittenWishCell.reuseId)
```

Now you know how to register a cell.

Point 5:

The groups we divide cells into are called sections. To create a new section we need to do the following:

- Override the **numberOfSections** method of the **UITableViewDataSource** protocol to return 2. Don't forget to add this number to the **Constants**.
- In **numberOfRowsInSection** return the amount of cells for each section. In our case, it is best to have a single **AddWishCell**. Use if or switch for section variable.
- Use if or switch in the **cellForRowAt** method. Use indexPath.section. In section 0 we should create AddWishCell, in section 1 we leave the code we wrote for WrittenWishCell.

Some of you might ask “Where or when did we write the AddWishCell?”. We indeed never did write it. This is a task for you to accomplish. You might want to follow instructions for point 6 to do it in one go. Good luck!

Point 6:

To add a wish to the table, we need AddWishCell to have a few things:
- A UITextView to input the wish.
- A Button to add the wish.

When the button is pressed the value of the **UITextView** (unless empty) is added as a wish to the wishArray. To do so we want an optional closure in the AddWish cell: **var addWish: ((String) -> ())?**. This allows us to change wishArray from inside the cell.

Once the wishArray is updated, do a **table.reloadData()**

Point 7:

To save wishes in **UserDefaults**, follow these steps:

Create a stored instance of the user defaults not to access the **standard** every time:

```
private let defaults = UserDefaults.standard
```

Every time you change the **wishArray** save it to the defaults:

```
defaults.set(stringArray, forKey: Constants.wishesKey)
```

On the **WishStoringViewController** loading, access the stored data and assign it to the **wishArray**:

```
defaults.array(forKey: Constants.wishesKey) as? [String]
```

Once you are finished, don't forget to remove the temporary wish.

Conclusion:

This tutorial guides you on how to use the **UITableView** and showcases the issues you might face working with table views. You do not see that much code anymore because you already know how to do some things. In the future, the amount of code provided in the tutorial will continue to decrease in direct correlation with your knowledge.

You can successfully finish this tutorial and get **7 points**. This is a good grade. To earn more, do the remaining points by learning and implementing something new.