**Домашнее задание к занятию «2.3. ООП: Композиция, наследование и интерфейсы»**

В качестве результата пришлите ссылки на ваши GitHub-проекты в личном кабинете студента на сайте [netology.ru](https://netology.ru).

**Важно**: ознакомьтесь со ссылками на главной странице [репозитория с домашними заданиями](https://github.com/netology-code/kt-homeworks/blob/master/README.md).

Если у вас что-то не получилось, то оформляйте Issue [по установленным правилам](https://github.com/netology-code/kt-homeworks/blob/master/report-requirements.md).

Нужно сделать все задачи в **одном** репозитории.

**Как сдавать задачи**

1. Создайте на вашем компьютере Gradle-проект.
2. Инициализируйте в нём пустой Git-репозиторий.
3. Добавьте в него готовый файл [.gitignore](https://github.com/netology-code/kt-homeworks/blob/master/.gitignore).
4. Добавьте в этот же каталог остальные необходимые файлы.
5. Сделайте необходимые коммиты.
6. Создайте публичный репозиторий на GitHub и свяжите свой локальный репозиторий с удалённым.
7. Сделайте пуш и удостоверьтесь, что ваш код появился на GitHub.
8. Ссылку на ваш проект отправьте из личного кабинета на сайте [netology.ru](https://netology.ru).
9. Необязательные задачи можно не сдавать — это не повлияет на получение зачёта. В этом ДЗ все задачи обязательные.

**Задача №1 - Посты (без Attachments)**

Мы с вами изучили Nullable, поэтому пришло время закрыть вопросы с постами: привыкайте к тому, что ваш код никуда не исчезает, и вам предстоит поддерживать то, что вы сами же когда-то написали (старая шутка "да кто вообще это писал?"). Поэтому мы в том числе будем отрабатывать и этот навык.

Доработайте первую задачу из предыдущей лекции так, чтобы ваша система отображала все поля объекта (кроме attachments), описанного в документации.

Напоминаем, что если у вас по каким-то причинам не доступен Vk, вы всегда можете воспользоваться сохранённой версией из каталога [assets](https://github.com/netology-code/kt-homeworks/blob/master/06_inheritance/assets)

Что мы хотим получить:

1. Data класс Post.
2. Внутри Post могут быть Nullable свойства.
3. WallService, который внутри себя хранит посты в массиве.

Итог: у вас должен быть репозиторий на GitHub, в котором расположен ваш Gradle-проект. Автотесты также должны храниться в репозитории.

Вы можете оформить это как Pull Request к предыдущему проекту (присылайте в личном кабинете ссылку на PR).

**Важно**: после ваших обновлений, функциональность WallService должна оставаться работоспособной (т.е. автотесты должны проходить).

**Задача №2 - Attachments**

Надеемся, что с постами было не так уж и сложно. А вот с [attachments](https://github.com/netology-code/kt-homeworks/blob/master/06_inheritance/assets/attachments.pdf) будет сложнее.

Почему? Потому что у вас в массиве будут храниться совершенно разные объекты (на самом деле не совсем так, но сейчас разберёмся).

Сам Vk их хранит вот так:

{

"attachments": [

{

"type": "photo",

"photo": {

"id": 1,

"album\_id": 1,

"owner\_id": 1,

"user\_id": 1

}

}, {

"type": "video",

"video": {

"id": 1,

"album\_id": 1,

"owner\_id": 1,

"user\_id": 1

}

}

]

}

Что это значит? Это значит, что у вас есть объект типа Attachment, в котором есть поле type, а вот какое второе поле у него есть - мы не знаем (оно определяется на базе значения поля type).

Соответственно, у вас два варианта:

1. Сделать Attachment интерфейсом.
2. Сделать Attachment абстрактным классом.

Тогда всё сходится и вы можете создать массив типа Attachment и через Smart Casts работать уже работать с конкретным полем.

Собственно, это вам и необходимо сделать.

Поскольку типов вложений очень много, вам достаточно выбрать для реализации любые 5.

Подсказка

Вы можете оформить это как Pull Request к задаче №1 (присылайте ссылку в личном кабинете на PR).

**Важно**: после ваших обновлений, функциональность WallService должна оставаться работоспособной (т.е. автотесты должны проходить).

**Задача №3 - Sealed классы\***

**Важно**: это не обязательная задача. Её выполнение не влияет на получение зачёта по ДЗ.

Одной из возможностей Kotlin для решения предыдущей задачи являются [Sealed ("запечатанные") классы](https://kotlinlang.org/docs/reference/sealed-classes.html).

Что это значит? Это значит, что вы можете определить фиксированную иерархию типов (ведь количество типов вложений у вас фиксировано).

Тогда в выражениях типа when вам не придётся писать else.

Нужно почитать [документацию](https://kotlinlang.org/docs/reference/sealed-classes.html) и сделать Pull Request с реализацией на базе Sealed классов (вы можете подглядеть подсказку).

Подсказка

Вы можете оформить это как Pull Request к задаче №1 (присылайте ссылку в личном кабинете на PR).

**Важно**: после ваших обновлений, функциональность WallService должна оставаться работоспособной (т.е. автотесты должны проходить).