Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №2
на тему «Дослідження роботи з компонентом FRAGMENT»
З дисципліни «Розробка мобільних застосувань під Android»

Виконала:
студентка 3 курсу
групи ІС-21
Бєльчик Софія

Київ-2025

Мета роботи — дослідити створення та взаємодію з компонентом Фрагмент (Fragment) компоненту Діяльність та набути практичні навички з використання фрагментів для інтерфейсу користувача.

Варіант - 14

Завдання:
Написати програму під платформу Андроїд, яка має інтерфейс, побудований з декількох фрагментів згідно варіанту. Перший фрагмент представляє з себе форму для введення даних та кнопку підтвердження («ОК»), а інший фрагмент відображає результат взаємодії. Тобто другий фрагмент містить тестове поле з результатом та кнопкою «Cancel» (якщо згідно варіанту така існує, якщо ж за варіантом її немає –можете додати за власним бажанням), яка очищає або приховує (або видаляє) другий фрагмент та очищає форму введення з першого фрагменту. Зверніть увагу, що робота з фрагментами відбувається в рамках однієї Діяльності.
Примітка: завдання відповідає варіанту лабораторної роботи №1.

14.	Вікно містить згорнутий список книг (автори), групу опцій (роки видання), тобто радіо-батони, та кнопку «ОК». Вивести інформацію щодо вибору.

Посилання гітхаб:https://github.com/SofiaBielchik/task2.git

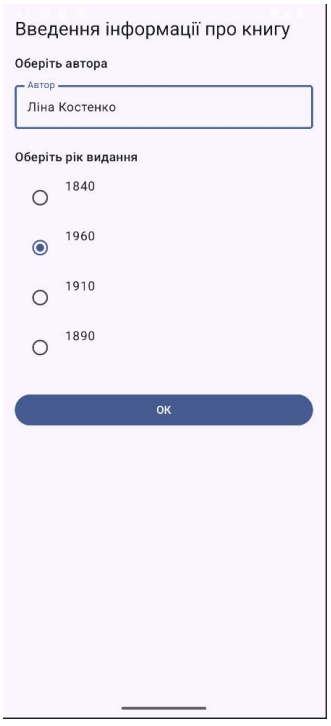

Рис. 1. Екран першого фрагмента з заповненими даними

- Користувач у полі AutoCompleteTextView ввів «Ліна Костенко».
- Обрано радіокнопку з текстом «1960».
- Поки що не було натискання «OK», тому ми бачимо форму введення.

Рис. 2. Екран другого фрагмента з результатом

- Після натискання «OK» у InputFragment ми перейшли на ResultFragment.
- У заголовку показано «Результат».
- Кнопка «Cancel» дозволяє повернутися до першого фрагмента та очистити дані.

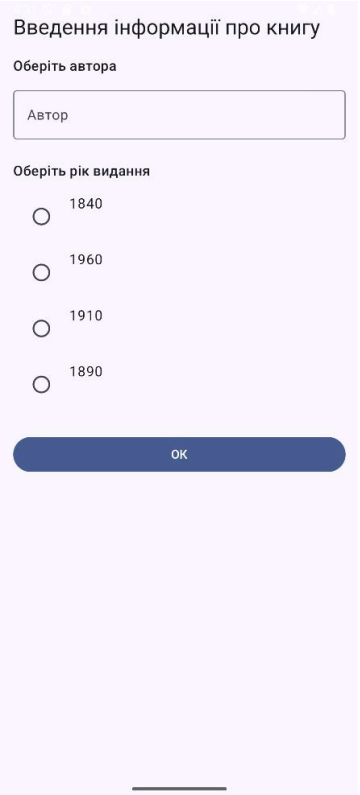

Рис. 3. Повернення до першого фрагмента (початковий стан)

- Користувач натиснув «Cancel» у другому фрагменті.
- Викликано метод clearInputs() з InputFragment, тому поля введення (AutoCompleteTextView) порожні, а жодна радіокнопка не обрана.
- Повернулися до початкового екрану першого фрагмента.

Висновки:

У другій лабораторній роботі ми

- Створили Android-застосунок із двома фрагментами: InputFragment і ResultFragment.
- Налаштовали динамічну заміну фрагментів у межах однієї MainActivity за допомогою FragmentManager та FragmentTransaction.
- Забезпечели передачу даних із першого фрагмента до другого через інтерфейс InputListener і Bundle.
- Реалізували повернення до форми введення та очищення полів за допомогою методу clearInputs() у InputFragment.
- Під час натискання «OK» без повного заповнення полів користувач побачить AlertDialog із проханням заповнити всі дані.

- Наведені скріншоти демонструють:
    - Перегляд заповненого першого фрагмента (Рис. 1).
    - Перегляд другого фрагмента з відображеним результатом (Рис.2).
    - Повернення до першого фрагмента в початковому (порожньому) стані (Рис. 3).

Таким чином, з’ясовано принцип роботи з фрагментами, їх динамічної заміни, передачі даних між ними та керування стеком транзакцій. Ці навички є базовими для створення масштабованих і гнучких Android-застосунків із багатокомпонентним інтерфейсом.

Контрольні запитання:

1. Призначення та можливості компоненту “Фрагмент”

Фрагмент (*Fragment*) — це частина інтерфейсу користувача (UI) та поведінки, яку можна вбудовувати в Activity. Основні можливості:

- Модульність: дозволяє розбити складний інтерфейс на автономні частини (фрагменти), кожен із яких має власний життєвий цикл. Повторне використання: той самий фрагмент можна використовувати в різних Activity або на різних екранах (наприклад, на планшеті один фрагмент показати поряд із іншим, а на телефоні — заміняти послідовно).
- Динамічна зміна UI: можна додавати, замінювати або видаляти фрагменти за допомогою FragmentManager без перезапуску Activity.
- Управління власним життєвим циклом: фрагмент має методи onCreateView(), onViewCreated(), onDestroyView() тощо, щоб контролювати створення/знищення в’ю-шаров.
- Взаємодія з Activity: через інтерфейси або ViewModel можна передавати дані між Activity та фрагментом або між різними фрагментами в одній Activity.

2. Життєвий цикл компонента “Фрагмент”

Життєвий цикл фрагмента схожий на життєвий цикл Activity, але має власні етапи:

1. onAttach(Context)

    - Фрагмент «прикріплюється» до Activity.
    - Тут можна отримати контекст Activity, перевірити, чи імплементує вона потрібні інтерфейси.

2. onCreate(Bundle)
    - Фрагмент створюється.
    - Тут зазвичай дістають аргументи із getArguments(), ініціалізують змінні, що не залежать від UI.

3. onCreateView(LayoutInflater, ViewGroup, Bundle)
    - “Надути” розмітку фрагмента з XML.
    - Повернути кореневий View, який стає в’ю-шаром фрагмента.

4. onViewCreated(View, Bundle) (опціонально)
    - Викликається одразу після onCreateView(), коли в’ю вже створені.
    - Тут виконують налаштування UI-елементів (наприклад, findViewById, налаштування кнопок).

5. onStart()
    - Фрагмент стає видимим.
    - Подібно до того, як onStart() у Activity.

6. onResume()
    - Фрагмент набуває фокусу і готовий до взаємодії з користувачем (нерідко тут підписуються на стріми/спостерігачі).

7. onPause()

    - Фрагмент втрачає фокус (наприклад, коли Activity переходить у фон або відбувається заміна іншим фрагментом).

8. onStop()
    - Фрагмент більше не відображається.

9. onDestroyView()

- Знищується в’ю-шар фрагмента (але сам об’єкт фрагмента все ще існує).
- Тут можна обнуляти посилання на UI-елементи, щоб уникнути витоку пам’яті.

10. onDestroy()
    - Фрагмент готується до вивантаження з пам’яті (звільняються невеликі ресурси, що не залежать від UI).

11. onDetach()
    - Фрагмент «від’єднується» від Activity.
    - Тут можна обнулити посилання на контекст та інтерфейси.

3. Способи створення компонента “Фрагмент”

1. Через XML (static add)

У розмітці Activity (activity_main.xml) напряму вказати:

```
<fragment
   android:id="@+id/myFragment"
   android:name="com.example.MyFragment"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

Такий фрагмент одразу додається під час інфляції Activity.
Мінус: не дозволяє динамічно змінювати фрагмент у контейнері без перезавантаження Activity.

2. Програмно (dynamic add/replace)

Використовуючи FragmentManager та FragmentTransaction у коді Activity:

```
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
ft.add(R.id.fragmentContainer, new MyFragment(), "TAG");
ft.commit();
```

- Або replace(...), remove(...), addToBackStack(...).
  Дозволяє динамічно керувати додаванням, видаленням і

заміною фрагментів в контейнері.

3. Через Navigation Component (Jetpack Navigation)

    - Описати nav_graph.xml із вершинами (фрагментами) і зв’язками (сейф-аргументи).
    - У Activity включити NavHostFragment.
    - Використовувати NavController для переходів: navController.navigate(R.id.nextFragment, bundle).
    - Дає змогу автоматично генерувати класи для безпечного передавання аргументів, а також керувати back stack.

4. Способи управління компонентом “Фрагмент”

1. FragmentManager & FragmentTransaction

    - Додати фрагмент: add(containerId, fragment, tag)
    - Замінити фрагмент: replace(containerId, fragment, tag)
    - Видалити фрагмент: remove(fragment)
    - Додати у back stack: addToBackStack(name)
    - Підтвердити зміни: commit(), commitAllowingStateLoss(), commitNow().

2. Використання ViewModel та LiveData

    - Фрагменти можуть отримувати спільний ViewModel, оголошений у межах Activity (new ViewModelProvider(requireActivity()).get(SharedViewModel.class)).
    - Через LiveData передавати дані між фрагментами без прямого виклику методів.

3. Navigation Component
    - Замість явних транзакцій, описуємо навігаційний граф, а потім виконуємо navController.navigate(...).
    - Стек транзакцій фрагментів автоматично керується Navigation Component.

4. Protocol-Based Communication (інтерфейси)
    - Як у цій роботі: перший фрагмент оголошує інтерфейс, Activity його імплементує, а потім передає повідомлення іншому фрагменту.

5. Способи взаємодії між фрагментами

1. Через інтерфейс, реалізований у Activity

    - Фрагмент А оголошує внутрішній інтерфейс OnDataPassListener.
    - Activity імплементує цей інтерфейс.
    - Фрагмент А викликає listener.onDataPass(data).
    - Activity отримує дані і передає їх у фрагмент В (через транзакцію або безпосередньо викликаючи метод фрагмента В).

2. Через спільний ViewModel

    - Activity створює/отримує спільний ViewModel (new ViewModelProvider(requireActivity())...).
    - Фрагмент А записує дані в LiveData у ViewModel.
    - Фрагмент В спостерігає цю LiveData і реагує на зміни.

3. Через Bundle/Аргументи при створенні фрагмента

При заміні фрагментів передати дані у Bundle:
```
Fragment fragmentB = new FragmentB();
Bundle args = new Bundle();
args.putString("key", value);
fragmentB.setArguments(args);
```

Потім у FragmentB:
```
String value = getArguments().getString("key");
```

6. Поняття “система”, “мала система” та “мобільна платформа”

1. Система

- Сукупність взаємопов’язаних елементів, які утворюють єдине ціле й взаємодіють між собою для досягнення певних цілей.
- Приклад: операційна система Android — комплекс програмних модулів (ядро Linux, бібліотеки, фреймворк, системні сервіси), які разом забезпечують роботу пристрою.

2. Мала система

    - Компактна за масштабом система, що працює на обмежених ресурсах (апаратних чи програмних).
    - Приклад: вбудована ОС, що керує побутовим приладом (наприклад, керування мотором пральної машини) — працює на мікроконтролері з обмеженим обсягом пам’яті, обчислювальною потужністю.

3. Мобільна платформа

    - Комплекс програмних компонентів, що дозволяє розробникам створювати програми для мобільних пристроїв.
    - Складається з операційної системи (Android, iOS), SDK (інструментів, бібліотек, API) та середовища розробки (Android Studio, Xcode).
    - Забезпечує:
        - Управління апаратними ресурсами (камера, GPS, сенсори);
        - Стандартизовані інтерфейси для UI (контейнери, компоненти керування);
        - Механізми збереження даних (SQLite, SharedPreferences, файли).

7. Типи мобільних застосунків

1. Натівні (Native apps)

    - Створені під конкретну мобільну платформу (наприклад, Java/Kotlin → Android; Swift/Objective-C → iOS).

- Мають безпосередній доступ до API та апаратних можливостей.
- Зазвичай дають найкращу продуктивність і UX, але для кожної платформи потрібно окремий код.

2. Гібридні (Hybrid apps)

   - Поєднують веб-технології (HTML5, CSS, JavaScript) з обгорткою-нативною оболонкою (наприклад, Apache Cordova, Ionic).
   - Програма фактично запускає локальний WebView, де завантажується веб-контент.
   - Дозволяють швидко розробляти під кілька платформ одночасно, але можуть мати обмеження в продуктивності та доступі до деяких апаратних можливостей.

3. Кросплатформенні (Cross-platform)

   - Використовують фреймворки, які дозволяють один код (часто на JavaScript/TypeScript або Dart) компілювати під різні платформи.
   - Приклади: React Native (JavaScript), Flutter (Dart), Xamarin (C#).
   - Дає змогу розробляти один код, що генерує нативні UI-компоненти (React Native, Xamarin) або власні рендер-пакети (Flutter), але одного разу треба налаштувати під кожну платформу.

4. Progressive Web Apps (PWA)

   - Веб-застосунки, які «пакуються» як мобільні застосунки.
   - Працюють у браузері або через механізм додавання на домашній екран, підтримують офлайн (service workers).
   - Не мають повного доступу до апаратних можливостей (залежить від браузера), але їх можна розповсюджувати безпосередньо через інтернет.

8. Класифікація та загальна характеристика середовищ розробки мобільних застосунків

1. Android Studio
    - Офіційне середовище розробки для Android (збудоване на IntelliJ IDEA).
    - Містить:
        - Android SDK (інструменти командного рядка, емулятори, бібліотеки).
        - Android Virtual Device (AVD) Manager для створення емуляторів.
        - Layout Editor з інтегрованим прев'ю макетів.
        - Gradle (система збірки).
        - Шаблони проектів (Activity, фрагмент, Service тощо).
    - Дозволяє розробляти як Java-, так і Kotlin-застосунки.

2. Xcode
    - Офіційне IDE для розробки під iOS/macOS/tvOS/watchOS. Містить Interface Builder (для будування UI), Instruments (для профілювання), Simulators, Swift Package Manager.
    - Використовується для створення програм на Swift або Objective-C.

3. Visual Studio

    - Переважно для кросплатформенної розробки з Xamarin.
    - Дозволяє писати один код (C#) і генерувати нативні UI Android/iOS.
    - Містить емульовані середовища та емулятор для Windows + Android.

9. Класифікація та загальна характеристика мобільних платформ

1. Android (Google)
    - Операційна система на ядрі Linux.
    - Найпоширеніша мобільна платформа у світі (понад 70 % ринку).

- Відкритий вихідний код (AOSP), дозволяє виробникам пристроїв модифікувати ОС.
- Підтримує Java та Kotlin (перетворює байткод у нативний ART code).
- Google Play Store — основний маркет, але існують альтернативні магазини (Amazon Appstore тощо).

2. iOS (Apple)

    - Закрита екосистема (Unix-подібна), виконується на пристроях Apple (iPhone, iPad, iPod Touch).
    - Підтримуються лише Objective-C та Swift.
    - App Store — єдина офіційна торгова платформа для розповсюдження додатків.

3. HarmonyOS (Huawei)

    - Власна ОС від Huawei на базі OpenHarmony (відгалуження AOSP).
    - Має модульну архітектуру, орієнтовану на кросплатформену взаємодію з IoT.
    - Підтримує розробку на Java, Kotlin, C/C++ (Huawei DevEco Studio).