ЛАБОРАТОРНА РОБОТА 3

Тема: Використання фрагментів у мобільних додатках

Фрагмент (fragments) є об'єктом контролера, якому активність може довірити виконання операцій. Найчастіше такою операцією є управління призначеним для користувача інтерфейсом - цілим екраном або його частиною. Фрагмент, який управляє інтерфейсом користувача, називається UI-фрагментом. UI-фрагмент має власне представлення, що заповнюється завдяки файлу макета. Представлення фрагменту містить елементи інтерфейсу, які призначено для користувача. Представлення активності має місце, в яке вставляється представлення фрагмента. Активність може управляти як одним фрагментом, так і може мати декілька місць для представлення декількох фрагментів. Фрагменти, що зв’язані з активністю, можуть використовуватись для формування та зміни екрану у відповідності до потреб додатку та користувачів, а представлення активності формально залишається незмінним протягом життєвого циклу. Для реалізації додатків типу “список — деталізація”, де деталізація є динамічною, фрагменти є одним із рішень.

Застосування UI-фрагментів дозволяє розділити призначений для користувача інтерфейс додатку на структурні блоки. У дійсності, це є корисним не тільки для додатків типу «список — деталізація», а і спрощує роботу з окремими блоками при створені інтерфейсів із вкладками, анімованих панелей та іншого. Але реалізація можливостей гнучкості інтерфейсу користувача підвищує складність та обсяги коду додатку.

Фрагмент є багаторазовим класом, що реалізує частину активності та частину інтерфейсу користувача, вбудований в активність і не може працювати незалежно від неї.

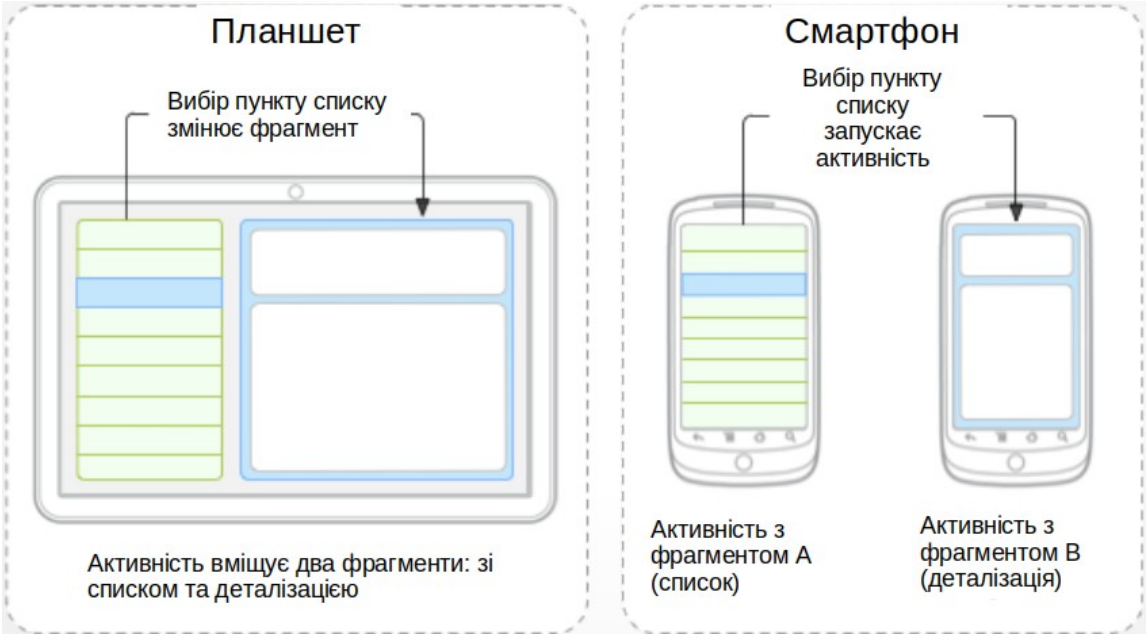


Рис. 1 — Поєднання фрагментів та активностей для відтворення GUI додатків.

Фрагмент є комбінацією файлу XML-макета та класу Java, подібно до активностей. Використання бібліотеки підтримки забезпечує фрагментам зворотню підтримку до всіх відповідних версій Android. Фрагменти інкапсулюють вигляд та логіку (View та Control), що полегшує їх повторне використання у рамках активності.

У рамках фрагментарно-орієнтованої архітектури активності стають навігаційними контейнерами, які головним чином відповідають за зв’язок з іншими активностями, через представлення фрагментів та передавання даних.

Для демонстрації використання фрагментів розробимо додаток Warehouse, в якому фрагменти будуть використовуватись для представлення та внесення нових записів про інвентар.

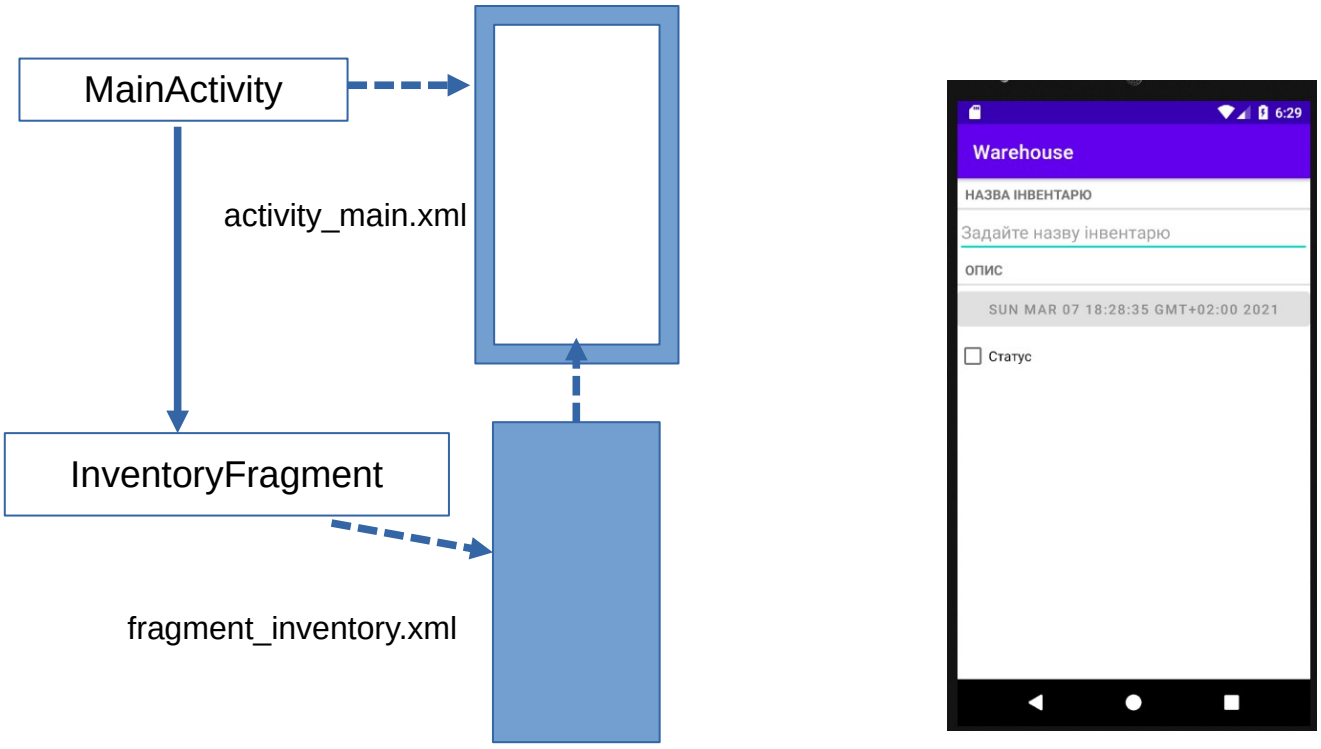
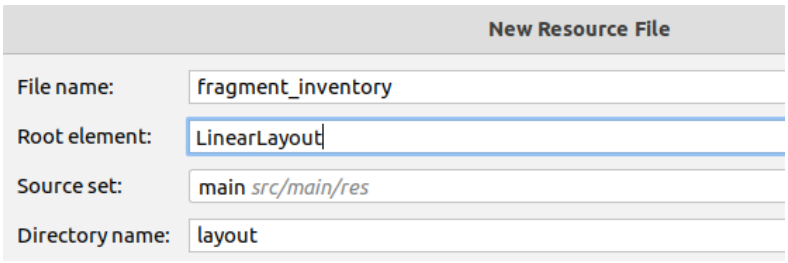


Рис. 2 — Загальний вигляд додатку та загальна архітектура його реалізації, де MainActivity — головна активність додатку, яка відіграє роль хосту для фрагменту, InventoryFragment — фрагмент, що реалізує GUI додатку та його контролер. Файли activity_main.xml та fragment_inventory.xml є макетами головної активності та фрагменту, відповідно.

1. Створюємо новий проект (обираємо Empty project). Фрагмент до додатку можна додати або статично — через явне посилання у макеті головної активності, або динамічно — через програмний код в MainActivity. Останній підхід є більш гнучким і надає більше можливостей до модифікації вигляду фрагменту. Тому вміст activity_main.xml змінюємо на:

```
<?xml version="1.0" encoding="utf-8"?>
    <FrameLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/fragment_container"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
    </FrameLayout>
```

2. Створюємо новий файл макету fragment_inventory.xml у теці res/layout. У вікні інструментів Project обираємо команду New → Layout Resource File, та задаємо відповідне ім’я:

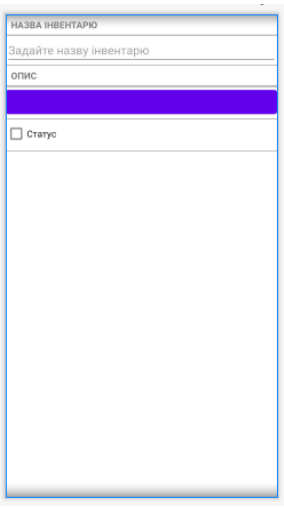


До макету додаємо необхідні віджети: TextView, EditText, TextView, Button та CheckBox. За результатом файл цього макету буде виглядати наступним чином:

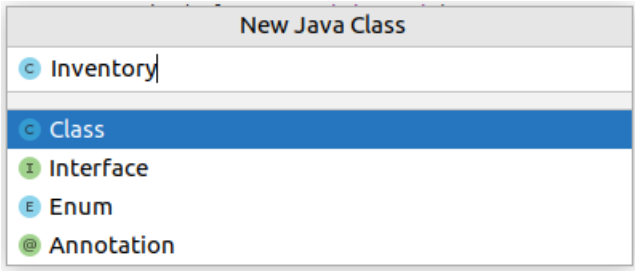
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/inventory_title_label"/>

    <EditText
        android:id="@+id/inventory_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/inventory_title_hint"/>

    <TextView
        style="?android:listSeparatorTextViewStyle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/inventory_details_label"/>

    <Button
        android:id="@+id/inventory_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>

    <CheckBox
        android:id="@+id/inventory_solved"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/inventory_solved_label"/>

</LinearLayout>
```

Для відображення необхідної інформації на віджетах додаємо до файлу текстових ресурсів res/values/strings.xml наступні значення:

```
<resources>    <string name="app_name">Warehouse</string>
    <string name="inventory_title_hint">Задайте назву інвентарю</string>
    <string name="inventory_title_label">Назва інвентарю</string>
    <string name="inventory_details_label">Опис</string>
    <string name="inventory_solved_label">Статус</string>
</resources>
```



3. Для додавання певного інвентарю, зберігання та модифікації інформації про нього, необхідно відповідний клас Inventory. У дереві проекту необхідно клацнути правою кнопкою миші, наприклад, на класі MainActivity та обрати у контекстному меню New → Java Class, задати ім’я нового класу та натиснути Enter:



Додамо до класу необхідні поля та методи:

```
public class Inventory {
    private UUID mId; // ідентифікатор інвентарю
    private String mTitle; // назва
    private Date mDate; // дата внесення
    private boolean mSolved; // статус

    public Inventory(){
        mId = UUID.randomUUID();
        mDate = new Date();
    }

    public UUID getmId() {
        return mId;
    }

    public String getmTitle() {
        return mTitle;
    }

    public void setmTitle(String mTitle) {
        this.mTitle = mTitle;
    }

    public Date getmDate() {
        return mDate;
    }

    public void setmDate(Date mDate) {
        this.mDate = mDate;
    }

    public boolean ismSolved() {
        return mSolved;
    }

    public void setmSolved(boolean mSolved) {
        this.mSolved = mSolved;
    }
}
```

4. Створюємо контролер - клас InventoryFragment. Його задача — виводити детальну інформацію про інвентар та надавати операції з її модифікації.

```
public class InventoryFragment extends Fragment {
    private Inventory mInventory;
    private EditText mTitleField;
    private Button mDateButton;
    private CheckBox mSolvedCheckBox;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
        mInventory = new Inventory();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                                                Bundle savedInstanceState){
        View view = inflater.inflate(R.layout.fragment_inventory, container, false);
        mTitleField = (EditText) view.findViewById(R.id.inventory_title);
        mTitleField.addTextChangedListener(new TextWatcher() {
            @Override
            public void beforeTextChanged(CharSequence s, int start, int count, int after) {
                // empty
            }

            @Override
            public void onTextChanged(CharSequence s, int start, int before, int count) {
                mInventory.setmTitle(s.toString());
            }

            @Override
            public void afterTextChanged(Editable s) {
                // empty
            }
        });

        mDateButton = (Button) view.findViewById(R.id.inventory_date);
        mDateButton.setText(mInventory.getmDate().toString());
        mDateButton.setEnabled(false);

        mSolvedCheckBox = (CheckBox) view.findViewById(R.id.inventory_solved);
        mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener(){
            @Override
            public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                mInventory.setmSolved(isChecked);
            }
        });
        return view;
    }
}
```

Створення та налаштування представлення фрагменту відбувається в методі життєвого циклу фрагмента `onCreateView`. У цьому методі заповнюється макет представлення фрагменту і заповнений об'єкт View повертається активному хосту. Перші два параметри методу необхідні для заповнення макету, а об'єкт Bundle використовується для відновлення представлення із збереженого. По-перше, зв'язується метод з відповідним layout:

```
View view = inflater.inflate(R.layout.fragment_inventory, container, false);
```

Також до класу додано об'єкти, що зв'язуються з віджетами на макеті фрагменту: `EditText mTitleField; Button mDateButton; CheckBox mSolvedCheckBox;`. Це дозволяє реалізувати деякі слухачі та контролювати параметри відповідних віджетів. Введення тексту у поле назви викликає метод `onTextChanged` у якому змінюється відповідний параметр об'єкту `mInventory` через виклик його методу `setmTitle()`.

5. Для того щоб в коді додати фрагмент до активності, необхідно явно звернутись з викликом до об'єкту FragmentManager активності. Компонент FragmentManager відповідає за управління фрагментами та додавання їх представлень до ієрархії представлень активності. FragmentManager керує двома структурами: списком фрагментів та стеком транзакцій фрагментів. По-перше створюється сам об'єкт FragmentManager:

```
FragmentManager fragmentManager = getSupportFragmentManager();
```

Для отримання екземпляру фрагменту певного класу, що його реалізує, необхідно звернутись до FragmentManager за ідентифікатором контейнерного представлення:

```
Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);
```

Якщо фрагмент за цим ідентифікатором існує, то його буде повернено, а якщо не існує, то його необхідно створити та зв’язати з ідентифікатором контейнерного представлення.

```
public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   FragmentManager fragmentManager = getSupportFragmentManager();
   Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);

   if(fragment == null){
     fragment = new InventoryFragment();
     fragmentManager.beginTransaction().add(R.id.fragment_container,fragment).commit();
   }
 }
}
```

Тепер додаток з однією активністю та фрагментом в ній можна запустити і перевірити його роботу.

7. Для надання більшої функціональності додатку реалізуємо два фрагменти, кожний з яких буде використовуватись у окремому хості (активності) та буде використовувати механізм інтентів для передавання даних між фрагментами у різних активностях (рис.3).

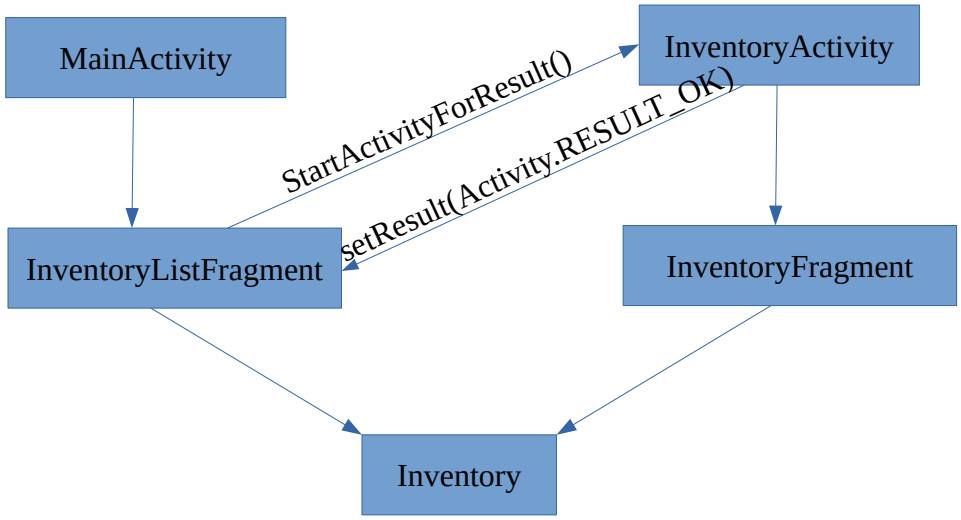


Рис. 3 — Загальна архітектура додатку. Чотири класи мають зв’язок з файлами макетів: MainActivity — activity_main.xml; InventoryListFragment — fragment_list_inventory.xml; InventoryActivity — activity_inventory.xml; InventoryFragment — fragment_inventory.xml.

Найменших змін у коді потребує клас MainActivity, тому що тепер фрагмент буде створюватись із класу InventoryListFragment:

```
public class MainActivity extends AppCompatActivity {

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
        FragmentManager fragmentManager = getSupportFragmentManager();
        Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);

        if(fragment == null){
          fragment = new InventoryListFragment();
          fragmentManager.beginTransaction().add(R.id.fragment_container, fragment).commit();
        }
    }
}
```

8. Клас `InventoryListFragment` буде відповідати за утримання списку інвентарю:

```
private ArrayList<Inventory> inventories;
```

підтримку списку інвентарю на інтерфейсі користувача через спеціально утворений адаптер InventoryAdapter, запуск другої активності InventoryActivity та обмін даними через інтенти. Цей клас розширює клас Fragment бібліотеки підтримки androidx.fragment.app.Fragment, що є більш універсальним підходом у порівнянні з бібліотеками певного API. Метод onCreate цього класу є достатньо простим і забезпечує відновлення стану фрагменту із попередньо збереженого та створення ArrayList inventories, який буде вміщувати записи інвентарю

```
      public class InventoryListFragment extends Fragment {
    private ArrayList<Inventory> inventories;

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        inventories = new ArrayList<>();
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        // зв'язуємо з відповідним макетом
        View view = inflater.inflate(R.layout.fragment_inventory_list, container, false);
        ListView listInventories = (ListView)view.findViewById(R.id.listInventory);
        // для формування структури списку використовуємо відповідний адаптер (див.нижче п.9)
        InventoryAdapter adapter = new InventoryAdapter(this.getContext(), inventories);
        listInventories.setAdapter(adapter);

        listInventories.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
            // тут будуть функції обробки при натисканні на елемент списку
            }
        });

        // формування інтенту та запуск активності при натисканні на кнопку додавання
        View fab = (View)view.findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(getActivity(), InventoryActivity.class);
                startActivityForResult(intent,0);
            }
        });
        return view;
    }

        // обробка зворотнього інтенту та додавання нового елементу до inventories
    @Override
    public void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == getActivity().RESULT_OK ) {
```

```
            Inventory inventory = new Inventory();
            inventory.setmTitle(data.getStringExtra("title"));
            inventory.setmDate((Date) data.getSerializableExtra("date"));
            inventory.setmSolved(data.getBooleanExtra("status",false));
            inventories.add(inventory);
        }
        else {
            Toast.makeText(getActivity(), "Wrong result", Toast.LENGTH_SHORT).show();
        }
    }
}
```

9. Для формування списків на інтерфейсі користувача створюємо адаптер InventoryAdapter та відповідний макет для нього `listview_layout_inventory.xml`.

Макет створює структуру кожного окремого запису, із яких потім складається список на інтерфейсі користувача в додатку. Тому зміна в цьому макеті веде до зміни інформації, що відображається на екрані. У нашому випадку використовуємо для списку два рядка — назву інвентарю та дату його створення (або реєстрації):

```
      <?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/inventory_title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="@color/black"/>
    <TextView
        android:id="@+id/inventory_date"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

В адаптері безпосередньо підключаємо макет та зв’язуємо певні значення параметрів його віджетів із значеннями елементів потрібного ArrayList:

```
      public class InventoryAdapter extends BaseAdapter {
    private Context context;
    private ArrayList<Inventory> inventories;

    public InventoryAdapter(Context context, ArrayList<Inventory> inventories){
        this.context = context;
        this.inventories = inventories;
    }

    @Override
    public int getCount() {
        return inventories.size();
    }

    @Override
    public Object getItem(int position) {
        return inventories.get(position);
    }

    @Override
    public long getItemId(int position) {
        return position;
    }
```

```
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        if(convertView == null){
            convertView =
LayoutInflater.from(context).inflate(R.layout.listview_layout_inventory,parent,false);
        }
        Inventory currentInventory = (Inventory) getItem(position);
        TextView textViewInventoryTitle = (TextView)
convertView.findViewById(R.id.inventory_title);
        textViewInventoryTitle.setText(currentInventory.getmTitle());
        TextView textViewInventoryDate = (TextView)
convertView.findViewById(R.id.inventory_date);
        textViewInventoryDate.setText(currentInventory.getmDate().toString());
        return convertView;
    }
}
```

10. До проекту додаємо нову активність InventoryActivity та створюємо макет для неї з ім’ям activity_inventory.xml. Цей макет достатньо простий і у даному проекті має той самий вміст, що і activity_main.xml:

```
      <?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</FrameLayout>
```

Головна його задача — надати контейнер для фрагменту другої активності. Макет зв’язуємо з класом в методі onCreate(). У цій активності буде використовуватись клас фрагменту, що працює з окремим елементом та його полями (див. пп. 1-6).

```
      public class InventoryActivity extends AppCompatActivity {

    Inventory inventory;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inventory);
        inventory = new Inventory();

        FragmentManager fragmentManager = getSupportFragmentManager();
        Fragment fragment = fragmentManager.findFragmentById(R.id.fragment_container);

        if(fragment == null){
          fragment = new InventoryFragment();
          fragmentManager.beginTransaction().add(R.id.fragment_container, fragment).commit();
        }
    }

    // додаємо до меню елемент типу “done”, який описано у файлі add_item.xml (див. далі п.10)
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.add_item, menu);
        return super.onCreateOptionsMenu(menu);
    }

    // реалізуємо функціональність при натисканні на певний елемент меню
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
```

```
        if (item.getItemId() == android.R.id.home) {
            finish();
            return true;
        }
    // при натисканні на елемент action_save (“done”) формується інтент у відповідь
        if (item.getItemId() == R.id.action_save) {
            InventoryFragment fragment = (InventoryFragment)
                      getSupportFragmentManager().findFragmentById(R.id.fragment_container);
            inventory = fragment.getmInventory();
            Intent intent = new Intent();
            intent.putExtra("title", inventory.getmTitle());
            intent.putExtra("id", inventory.getmId());
            intent.putExtra("date", inventory.getmDate());
            intent.putExtra("status", inventory.ismSolved());
            setResult(Activity.RESULT_OK,intent);
            finish();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    public void setInventory(Inventory inventory) {
        this.inventory = inventory;
    }
}
```

Зверніть увагу, що в методі onCreate() фрагмент створюється як екземпляр батьківського класу Fragment. Таким чином, цей екземпляр буде мати від класу InventoryFragment тільки ті поля та методи, які оголошено у класі Fragment.

Метод onOptionsItemSelected() обробляє натискання кнопок меню. Тому, при натисканні кнопки “action_save” (done) необхідно створити intent для відповіді, повідомити про результат менеджер активностей та завершити поточну активність. Для формування intent до нього необхідно додати параметри екземпляру inventory, який було створено у поточному фрагменті. Саме тому в цьому методі фрагмент створюється як екземпляр класу InventoryFragment, в якому реалізовано метод, що повертає екземпляр inventory.

10. Додаємо до ресурсів файл елементу меню add_item.xml з наступним вмістом:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">

    <item
        android:id="@+id/action_save"
        android:title="Зберегти"
        android:icon="@drawable/ic_done"
        app:showAsAction="ifRoom"/>
</menu>
```

11. Клас InventoryFragment подібний до п.4 і доповнюється методом `getmInventory()` для повернення екземпляру `mInventory` , що утворюється в цьому фрагменті.

```
public class InventoryFragment extends Fragment {
    private Inventory mInventory;
..................................

    @Override
    public void onCreate(Bundle savedInstanceState) {
.................................................
        mInventory = new Inventory();
    }
```

```
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                                        Bundle savedInstanceState){
.....................................................................................
        mTitleField.addTextChangedListener(new TextWatcher() {

.....................................................................................
            @Override
           public void onTextChanged(CharSequence s, int start, int before, int count){
                mInventory.setmTitle(s.toString());
            }
.....................................................................................
        });
.....................................................................................
        mDateButton.setText(mInventory.getmDate().toString());
.....................................................................................
        mSolvedCheckBox.setOnCheckedChangeListener(new OnCheckedChangeListener(){
           @Override
           public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
                mInventory.setmSolved(isChecked);
            }
        });
        return view;
    }

    public Inventory getmInventory() {
        return mInventory;
    }
}
```

Тепер додаток здатний створювати нові записи про інвентар і використовує для цього техніку фрагментів.

Практичне завдання

1. Реалізуйте приклади, наведені в теоретичній частині лабораторної роботи. До звіту включіть скриншоти.
2. Реалізуйте у додатку можливість продивлятись та корегувати вже існуючі записи.
3. Додайте можливість видаляти записи.
4. Наведіть у звіті програмний код реалізацій завдань 2 та 3, а також скриншоти інтерфейсів додатку.

Література

1. Phillips B. Android Programming: The Big Nerd Ranch Guide / B. Phillips, Ch. Stewart, K. Marsicano: 3rd Edition. — Big Nerd Ranch, 2017. — 720 p.
2. Android Studio // https://developer.android.com/studio/