ЛАБОРАТОРНА РОБОТА 2

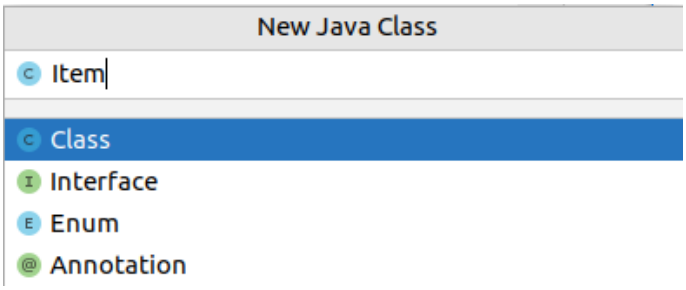
Тема: Android-додатки з декількома активностями

Декілька активностей дозволяють розширити функціональні можливості додатка та покращити його сприйняття користувачем. Розглянемо використання другої активності на прикладі додатку, що створює та зберігає деякий список, деталізація пунктів якого задається, виводиться та редагується за допомогою другої активності.

1. Створюємо новий проект (обираємо Empty project). Для реалізації списку на головному інтерфейсі додамо до activity_main.xml

```
<ListView
    android:id="@+id/listItems"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

2. Для зберігання інформації про пункти списку створюємо клас Item. У дереві проекту необхідно, наприклад, клацнути правою кнопкою миші на класі MainActivity та обрати у контекстному меню New → Java Class, задати ім’я нового класу та натиснути Enter:



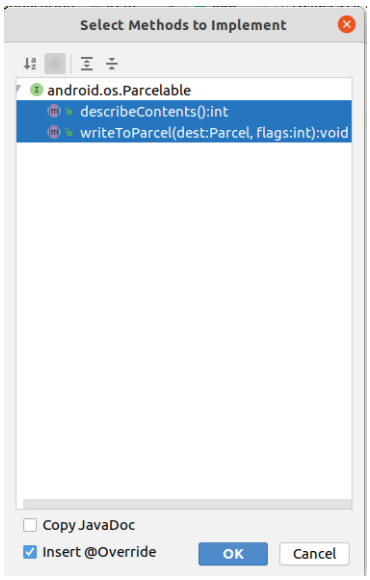
Додамо до класу наступні поля:

```
public class Item {
    String itemName = "";
    String itemClassification = "";
    String itemSize = "";
}
```

Для зберігання та відновлення даних екземплярів цього класу, а також зручного передавання між активностями зробимо цей клас, як той, що реалізує спеціальний інтерфейс Parcelable. Для цього додамо наступне:

```
public class Item implements Parcelable
```

Android Studio підкреслює цей рядок тому, що вимагає визначеності — або цей клас абстрактний, або необхідна імплементація методів інтерфейсу. Для автоматичного додавання необхідних методів натиснемо Alt+Enter , що викличе наступне діалогове вікно:



та натиснемо Enter для додавання методів до класу:

```
public class Item implements Parcelable{
    String itemName = "";
    String itemClassification = "";
    String itemSize = "";

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {

    }
}
```

Але Android Studio додала не все необхідне і тому ім’я класу позначено червоним. Позиціюємось на імені класу та натискаємо Alt+Enter, обираємо імплементацію методів Parcelable та тиснемо Enter. Як результат отримаємо наступне:

```
import android.os.Parcel;
import android.os.Parcelable;

public class Item implements Parcelable{
    String itemName = "";
    String itemClassification = "";
    String itemSize = "";

    protected Item(Parcel in) {
        itemName = in.readString();
        itemClassification = in.readString();
        itemSize = in.readString();
    }

    public static final Creator<Item> CREATOR = new Creator<Item>() {
        @Override
        public Item createFromParcel(Parcel in) {
            return new Item(in);
        }

        @Override
```

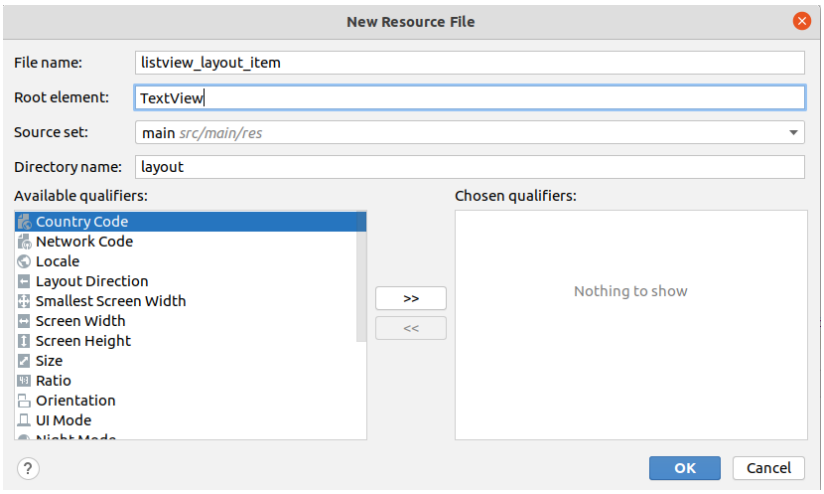
```
        public Item[] newArray(int size) {
            return new Item[size];
        }
    };

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeString(itemName);
        dest.writeString(itemClassification);
        dest.writeString(itemSize);
    }
}
```

Функція writeToParcel зберігає усі поля у спеціальне сховище, яке надає система Android, а конструктор дозволяє зчитувати ці поля при створені екземпляру класу, якщо вони були попередньо збережені.

3. Створюємо у класі MainActivity масив для пунктів списку:

```
private ArrayList<Item> items;
```

Для виводу елементу до списку необхідно підготувати розмітку елемента, для чого створюємо файл listview_layout_item.xml. Розгортаємо теку res, клацаємо правою кнопкою по теці layout та обираємо пункт New → Layout Resource File. У вікні New Resource File задаємо наступне:



Android Studio генерує файл розмітки, який модифікуємо 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"
    android:padding="12dp">
    <TextView
        android:id="@+id/item_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="16sp"
        android:textColor="#000000"/>
    <TextView
```

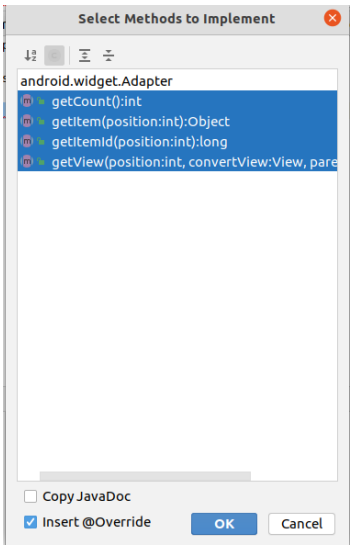
```
        android:id="@+id/item_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

Текстове поле с більшим за розміром текстом item_name буде вміщувати назву, а друге поле item_group – його класифікацію.

4. Для роботи з об’єктами створимо спеціальний адаптер на базі класу-основи для адаптерів BaseAdapter, у якому перевизначимо деякі методи. Створюємо клас ItemAdapter і додаємо посилання на батьківський клас BaseAdapter:

```
public class ItemAdapter extends BaseAdapter
```

Тепер додаємо імплементацію абстрактних методів:



і додаємо необхідну реалізацію методам:

```
public class ItemAdapter extends BaseAdapter {
    private Context context;
    private ArrayList<Item> items;

    public ItemAdapter(Context context, ArrayList<Item> items){
        this.context = context;
        this.items = items;
    }

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

    @Override
    public Object getItem(int position) {
        return items.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_item,parent,false);
        }
        Item currentItem = (Item) getItem(position);
        TextView textViewItemName = (TextView)
                                            convertView.findViewById(R.id.item_name);
        textViewItemName.setText(currentItem.itemName);
        TextView textViewItemClassification = (TextView)
                                            convertView.findViewById(R.id.item_group);
        textViewItemClassification.setText(currentItem.itemClassification);
        return convertView;
    }
}
```

Створений адаптер ItemAdapter буде застосовуватись у методі onCreate класу MainActivity для формування списку:

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

    items = new ArrayList<Item>();
    listItem = (ListView)findViewById(R.id.listItems);
    ItemAdapter adapter = new ItemAdapter(this,items);
    listItem.setAdapter(adapter);
//....
}
```

Таким чином створюється адаптер `ItemAdapter` та привласнюється елементу списку. Для обробки натискання на елемент списку зазвичай використовується обробник, шаблон якого виглядає наступним чином:

```
listItem.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
          //.........................
    }
});
```

На цьому етапі загальну архітектуру програми створено і слід перевірити можливість її роботи.

5. Для додавання нового запису до списку у мобільних додатках часто використовується спеціальна кнопка FloatingActionButton (кнопка, що плаває). Для додавання цього елементу насамперед слід перевірити, що Gradle має посилання на відповідну бібліотеку. Для цього в дереві проекту слід відкрити файл build.gradle (Module) і в розділі залежностей повинна знаходитись бібліотека:

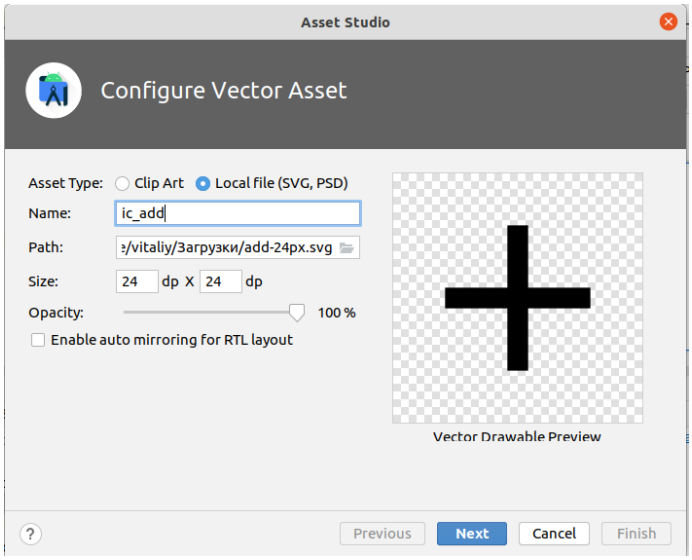
```
dependencies {
.......
    implementation 'com.google.android.material:material:1.3.0'
.......
}
```

Якщо такого запису немає, то його слід додати та натиснути кнопку Sync Now для оновлення проекту.

Додаємо елемент такої кнопки до розмітки головної активності:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:src="@drawable/ic_add"
    android:tint="@color/design_default_color_on_primary"
    android:layout_margin="16dp"/>
```

Параметр src визначає рисунок, що буде відображатись на кнопці, а tint – колір. Скоріш за все посилання на рисунок є неактивним (редактор позначає його червоним кольором). Зображення для кнопки можна взяти з Інтернет ресурсу https://material.io/resources/icons/. Значки слід брати у векторному форматі SVG. Для додавання векторного зображення до проекту до теки res → drawable додається New → Vector Asset. У вікні додавання зображення вказуємо місце знаходження відповідного ресурсу:



Після додавання цього знаку активізується посилання на нього в activity_main.xml.
Шаблон обробника цієї кнопки виглядає наступним чином:

```
View fab = (View)findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

і розміщується у MainActivity в методі onCreate.

6. Додаємо до проекту другу активність (File → New → Activity → EmptyActivity) та задаємо їй ім’я AddItemActivity. Змінюємо шаблон розмітки на LinearLayout (vertical), а до нього додаємо:

```
<com.google.android.material.textfield.TextInputLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/item_title_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textCapSentences"
        android:hint="Назва пункту"/>
</com.google.android.material.textfield.TextInputLayout>
```

У цій частині використовуються елементи управління TextInputLayout та TextInputEditText,

які є більш сучасними аналогами звичайного EditText та візуально краще відповідають сучасному стилю. Тому що певний пункт Item крім назви також має ще два поля — Клас та Розмір, то слід додати відповідні поля для введення подібно представленому вище коду розмітки. Таким чином, буде наступний вигляд:

Друга активність часто грає другорядну, допоміжну роль, тому користувачу необхідно надати можливість повертатись до головної активності. Тому у вікні другої активності повинні бути два елементи управління: один - для закривання активності без збереження даних, другий — із збереженням. Для скасування, зазвичай, використовується стрілка “<” , тому активуємо її в методі onCreate:

```
getSupportActionBar().setDisplayHomeAsUpEnabled(true);
```

Для створення кнопки з позитивним завершенням діалогу можна використати верхню смужку ActionBar. Додамо кнопку з рисунком Done, за допомогою якої будуть зберігатись дані та завершуватись діалог. Для створення меню необхідно створити файл розмітки, для чого необхідно додати до теки res New→AndroidResourceFile, задати ім’я: add_item.xml та вказати тип ресурсу – Menu. Android Studio створює теку menu та згенерує у ній файл меню, до якого додаємо наступний код:

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

та отримуємо наступний вигляд:

Далі вносимо зміни до класу активності, в якій буде виводитись меню. Програма повинна сконвертувати створений ресурс меню до програмного об’єкту. Для цього існує спеціальний метод MenuInflater.inflate(), який викликається у спеціальному методі зворотнього виклику onCreateOptionsMenu(). Цей метод призначено для виводу меню при натисканні кнопки **MENU** на пристрої:

```
@Override
public boolean onCreateOptionsMenu(Menu menu)
{
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.add_item, menu);
    return  super.onCreateOptionsMenu(menu);
}
```

При обрані одного з пунктів, а також при натисканні стрілки повернення, викликається метод onOptionsItemSelected:

```
  @Override
    public boolean onOptionsItemSelected(MenuItem item){
        if (item.getItemId() == android.R.id.home) {
            finish();
            return true;
        }
        if (item.getItemId() == R.id.action_save) {
// Тут буде зберігання даних
// ...
            finish();
            return true;
        }
        return super.onOptionsItemSelected(item);
    }
```

7. Для використання другої активності необхідно організувати її виклик та передавання даних. Для цього у головній активності MainActivity у методі onCreate в обробнику натискання плавучої кнопки додамо код:

```
fab.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(MainActivity.this, AddItemActivity.class);
        startActivityForResult(intent,0);
    }
});
```

У цьому коді створюється об’єкт класу Intent, який описує певні дії, а саме вказує клас активності, яку необхідно запустити. Після вказується метод startActivityForResult , що запускає активність. Також друга активність має запускатись при торкані елементу списку. Доповнимо цей код:

```
listItem.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Intent intent = new Intent(MainActivity.this, AddItemActivity.class);
        intent.putExtra("index",position);
        intent.putExtra("item", items.get(position));
        startActivityForResult(intent,0);
    }
});
```

У цьому коді створюється об’єкт Intent, в який ще додатково розміщують дані за допомогою методу putExtra : індекс обраного об’єкту в масиві, для можливості зміни даних у ньому на відредаговані користувачем, а також сам об’єкт з інформацією про item. Для отримання даних друга активність запитує в системі об’єкт Intent , за допомогою якого її було запущено. Доповнюємо код методу onCreate активності AddItemActivity наступним:

```
private int index = 0;
private Item item;
private TextInputEditText item_title_name;
private TextInputEditText item_title_class;
private TextInputEditText item_size;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_item);
    Intent intent = this.getIntent();
    index = intent.getIntExtra("index",-1);
    item = (Item) intent.getParcelableExtra("item");
    item_title_name = (TextInputEditText) findViewById(R.id.item_title_name);
    item_title_name.setText(item.itemName);
    item_title_class = (TextInputEditText) findViewById(R.id.item_title_class);
```

```
        item_title_class.setText(item.itemClassification);
        item_size = (TextInputEditText) findViewById(R.id.item_size);
        item_title_class.setText(item.itemSize);
    }
```

Де за допомогою метода getIntExtra отримується значення індексу, а за допомогою getParcelableExtra – отримується сам об’єкт. Отриманими даними заповнюються поля вводу.

8. При натисканні кнопки збереження дані необхідно взяти з полів вводу і зберегти їх у об’єкт Intent :

```
if (itemMenu.getItemId() == R.id.action_save) {
    // Тут буде зберігання даних
    item = new Item();
    item.itemName = ((TextInputEditText)
                                findViewById(R.id.item_title_name)).toString();
    item.itemClassification = ((TextInputEditText)
                                findViewById(R.id.item_title_class)).toString();
    item.itemSize = ((TextInputEditText) findViewById(R.id.item_size)).toString();
    Intent intent = new Intent();
    intent.putExtra("index", index);
    intent.putExtra("item", item);
    setResult(Activity.RESULT_OK,intent);
    return true;
}
```

На головній активності необхідно додати обробку результатів виклику активності. Для цього слугує метод onActivityResult, який перевизначається та викликається при завершені виклику будь-якої активності:

```
  @Override
    protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        super.onActivityResult(requestCode, resultCode, data);
        if (resultCode == RESULT_OK) {
             int index = data.getIntExtra("index", -1 );
             Item item = (Item) data.getParcelableExtra("item");
             if(index != -1){
                 items.set(index, item);
             }else{
                 items.add(item);
                 ListView listView = (ListView) findViewById(R.id.listItems);
                 ItemAdapter adapter = new ItemAdapter(this, items);
                 listView.setAdapter(adapter);
                 adapter.notifyDataSetChanged();
             }
        }
        else {
            Toast.makeText(this, "Wrong result", Toast.LENGTH_SHORT).show();
        }
    }
}
```

Якщо результатом виклику активності є код RESULT_OK , то отримуємо дані, що передаються другою активністю. А якщо індекс не задано (дорівнює -1), то дані додаються у кінець списку, а якщо задано – то замінюється об’єкт з вказаним індексом.

Після цього адаптер списку повідомляється про те, що дані змінено і потрібно зробити оновлення на екрані, що реалізується за допомогою методу notifyDataSetChanged.

Практичне завдання

1. Реалізуйте приклад, наведений в теоретичній частині лабораторної роботи. До звіту включіть скриншоти додатку, що працює в емуляторі.
2. У наведеному прикладі при зміні конфігурації пристрою його поточний стан не зберігається, тому всі зроблені записи втрачаються, наприклад, при повертанні пристрою. Використайте методи життєвого циклу головної активності для збереження поточного стану пристрою. У звіті наведіть реалізацію цих методів.
3. На основі наведеного прикладу розробіть додаток для старости групи, за допомогою якого можна додавати студентів у список, що відображається у головній актиності, а усі дії по виводу, додаванню або редагуванню інформації про студентів зі списку виконуються у другій активності.
4. Наведіть у звіті програмний код реалізації завдання 2 та скриншоти інтерфейсів додатку, який виконується у одному із емуляторів.

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/