МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

КАФЕДРА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Звіт

з практичної роботи № 3 з дисципліни

Програмування під Android

Виконала: Перевірив:

ст. гр. ПЗПІ-23-5 ст. викладач кафедри ПІ

Євсєєва Олександра Сокорчук Ігор Петрович

Харків 2024

ЗАВДАННЯ

Ознайомитися з основними типами діалогових панелей в Android.

Навчитися створювати та використовувати діалогові панелі в Android-застосунках.

Засвоїти методи для налаштування діалогових панелей та їх взаємодії з користувачем.

Засвоїти концепцію черги повідомлень (Message Queue) в Android.

Вивчити роботу з обробником повідомлень (Handler) для взаємодії з потоками.

Навчитися створювати та використовувати Handler для асинхронної обробки даних.

Навчитися працювати із компонентом RecyclerView для створення списків із прогортанням в Android, а також ознайомитися їх з принципами створення та використання адаптерів (Adapters) і керування даними у списках.

ХІД РОБОТИ

Cтворюємо новий проєкт та обираємо Empty Views Activity.

Діалогові панелі (Dialog) – це невеликі вікна, які з'являються на екрані поверх поточного інтерфейсу, щоб привернути увагу користувача або запросити в нього додаткові дії.

Типи діалогових панелей:

* AlertDialog: діалогове вікно з можливістю вибору між кількома варіантами або підтвердження певної дії.
* DatePickerDialog: дозволяє користувачу вибрати дату.
* TimePickerDialog: дозволяє користувачу вибрати час.
* CustomDialog: діалог з індивідуальним дизайном.
* ProgressDialog діалог із прогресом.

Почнемо із AlertDialog: додаємо кнопку для виклику діалогу:

1.<Button  
2. android:id="@+id/alert"  
3. android:layout\_width="wrap\_content"  
4. android:layout\_height="wrap\_content"  
5. android:text="Alert dialog"  
6. app:layout\_constraintStart\_toStartOf="parent"  
7. app:layout\_constraintTop\_toTopOf="parent"  
8. app:layout\_constraintEnd\_toEndOf="parent"/>

Ініцалізуємо та підписуємо подію за допомогою setOnClickListener у MainActivity:

1.alert.setOnClickListener(v ->  
2. new AlertDialog.Builder(context)  
3. .setTitle("Alert dialog")  
4. .setMessage("Alert dialog test")  
5. .setPositiveButton("Ok", (dialog, which) -> {  
6. Toast.*makeText*(context, "Alert dialog: okay", Toast.*LENGTH\_LONG*).show();})  
7. .setNegativeButton("Cancel", ((dialog, which) -> {  
8. Toast.*makeText*(context, "Alert dialog: cancel", Toast.*LENGTH\_LONG*).show();  
9. }))  
10. .create()  
11. .show());

Далі по списку DatePickerDialog. Аналогічно свторюємо кнопку із id date, ініціалізуємо та підписуємось на події із setOnClickListener із лямбдою. У виклику використовуємо допоміжний клас Calendar для визначення актуальної дати. Цей діалог дозволяє обрати дати у зручному вікні вибору, схожий на звичайний календар:

1.date.setOnClickListener(v -> {  
2. final Calendar calendar = Calendar.*getInstance*();  
3. new DatePickerDialog(context, (view, year, month, dayOfMonth) -> {  
4. Toast.*makeText*(context, String.*format*("year: %d, month: %d, day: %d", year, month, dayOfMonth), Toast.*LENGTH\_LONG*).show();  
5. }, calendar.get(Calendar.*YEAR*), calendar.get(Calendar.*MONTH*), calendar.get(Calendar.*DAY\_OF\_MONTH*))  
6. .show();  
7. });

Переходимо до TimePickerDialog. Аналогічно свторюємо кнопку із id time, ініціалізуємо та підписуємось на події із setOnClickListener із лямбдою. У виклику використовуємо допоміжний клас Calendar для визначення актуальної дати. Цей діалог дозволяє обрати час у зручному вікні вибору, схожий на звичайний годинник:

1.time.setOnClickListener(v -> {  
2. final Calendar calendar = Calendar.*getInstance*();  
3. new TimePickerDialog(context, (view, hourOfDay, minute) -> {  
4. Toast.*makeText*(context, String.*format*("hours: %d, minutes: %d", hourOfDay, minute), Toast.*LENGTH\_LONG*).show();  
5. }, calendar.get(Calendar.*HOUR\_OF\_DAY*), calendar.get(Calendar.*MINUTE*), true)  
6. .show();  
7. });

Після підтвердження вибору на викликаному діалогом view спливає повідомлення із обраним часом та датою відповідно. Це створено за допомогою елемента Toast.

Далі CustomDialog. Він трохи відрізняється від інших тим, що має особливе view, що можна модифікувати самому. Якщо більш конкретніше, то можна зробити власну розмітку для конкретного діалогу. Отже, спочатку створимо розмітку dialog\_custom.xml, в який помістимо напис «This is а custom dialog

» та дві кнопки для виходу із діалоги з позитивним та негативним результатами [ДОДАТОК А].

Обов’язково вказуємо, яку розмітку використовувати:

1. dialog.setContentView(R.layout.dialog\_custom);

Наступне ProgressDialog. Нехай такий діалог є застарілим, але про нього також необхідно сказати:

1.progress.setOnClickListener( v -> {  
2. ProgressDialog dialog = new ProgressDialog(context);  
3. dialog.setProgressStyle(ProgressDialog.*STYLE\_SPINNER*);  
4. dialog.setMessage("In progress...");  
5. dialog.create();  
6. dialog.show();  
7. });

У коді описано створення та налаштування діалогу: його стиль та текстове наповнення.

Із діалогами закінчили. Перейдемо до обробки Handlers.

Handler – це об’єкт, який дозволяє відправляти та обробляти повідомлення та виконувані об’єкти (Runnable) в потоці, пов’язаному з ним.

MessageQueue – черга, яка зберігає повідомлення та Runnable, що будуть оброблені об'єктом Handler.

Looper – це механізм, що циклічно обробляє чергу повідомлень для потоку.

У практичній я лише імітувала роботу у потоках, оскільки їхня реалізація могла б вийти набагато складніше. Подивимось на реалізацію виконання деякої дії із затримкою на 2 секунди, що виконується після натискання кнопки із id «handler»:

1.findViewById(R.id.*handler*).setOnClickListener(v -> {  
2. new Handler(Looper.*getMainLooper*()).postDelayed(() -> {  
3. Toast.*makeText*(context, "2 seconds after", Toast.*LENGTH\_LONG*).show();  
4. }, 2000);});

Так само відгук користувачу в інтерфейс відсилається за допомогою Toast.

Переходимо до роботи із RecyclerView.

* RecyclerView – основний контейнер для відображення списків або сіток елементів.
* LayoutManager – керує розташуванням елементів у списку (лінійний, сітковий або кастомний макети).
* Adapter – клас, який адаптує дані для відображення в RecyclerView.
* ViewHolder – зберігає посилання на елементи інтерфейсу для повторного використання.

Почнемо із додавання RecyclerView до файлу розмітки:

1. <androidx.recyclerview.widget.RecyclerView

2. android:id="@+id/list"

3. android:layout\_width="0dp"

4. android:layout\_height="0dp"

5. tools:listitem="@layout/item"

6. tools:itemCount="10"

7. android:layout\_marginStart="4dp"

8. android:layout\_marginEnd="4dp"

9. app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"

10. app:layout\_constraintTop\_toBottomOf="@id/handler"

11. app:layout\_constraintStart\_toStartOf="parent"

12. app:layout\_constraintEnd\_toEndOf="parent"

13. app:layout\_constraintBottom\_toBottomOf="parent"/>

Одразу видно, що використовуємо LinearLayoutManager для відображення компонентів лінійно. Звичайно, можна задати сітковий менеджер або будь-який створений самостійно. Самі компоненти, що відображаються називаються item.

В основному файлі коду генеруємо наповнення списку та викликаємо сам RecyclerView:

1.ArrayList<String> list = new ArrayList<>();  
2.for(int i = 0; i < 100; i++) {  
3. list.add("ItemAdapter item with id: ");  
4.}  
5.  
6.((RecyclerView) findViewById(R.id.*list*)).setAdapter(new ItemAdapter(list));

Створюємо ItemAdapter, окремий клас для того, щоб пояснити списку, як показувати елементи у ньому [ДОДАТОК Б].

Першим кроком створюємо onCreateViewHolder:

1.public ItemViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
2. View v;  
3. if (viewType == 1) {  
4. v = LayoutInflater.*from*(parent.getContext())  
5. .inflate(R.layout.*item*, parent, false);  
6. } else {  
7. v = LayoutInflater.*from*(parent.getContext())  
8. .inflate(R.layout.*item\_grey*, parent, false);}  
9. return new ItemViewHolder(v);}

Та створюємо ItemViewHolder:

1.public static class ItemViewHolder extends RecyclerView.ViewHolder {  
2. TextView id;  
3. TextView text;  
4. public ItemViewHolder(@NonNull View itemView) {  
5. super(itemView);  
6. id = itemView.findViewById(R.id.*id*);  
7. text = itemView.findViewById(R.id.*text*);  
8. }}

Цима двома кроками було створено каркас для списку. Тобто, тепер елемент знає що і як треба помістити у блок списку. Тепер напишемо, як саме треба наповнювати компоненти item:

1.public void onBindViewHolder(@NonNull ItemViewHolder holder, int position) {  
2. holder.id.setText(""+position);  
3. holder.text.setText(\_list.get(position) + position);}

В onCreateViewHolder зазначено тип view, за яким визнається яку саму розмітку програма буде використовувати у тому чи іншому випадку. Я зробила одну із двох панельок списку сірою. Визначається за подільністю на 2, отже, кожна друга плашка буде сірою. Самі компоненти мають 2 рядки: id та напис [ДОДАТОК В].

ВИСНОВКИ

Під час практичної роботи було створено додаток для ознайомлення та опановування основних типів діалогових панелей та їхні атрибути налаштування; засвоєння концепції черги повідомлень, оброки подій Handler; навчання працювати із з компонентом RecyclerView для створення списків із прогортанням в Android та використання адаптерів (Adapters), керування даними у списках.

Відеозвіт на YouTube. URL: <https://youtu.be/KrImQwyW8TU>

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Документація AndroidDevelopers dialogs. URL: <https://developer.android.com/develop/ui/views/components/dialogs> (дата звернення: 02.11.2024)
2. Документація AndroidDevelopers handlers. URL: <https://developer.android.com/reference/android/os/Handler> (дата звернення: 02.11.2024)
3. Питання Stack overflow «Why use handlers?» URL: <https://stackoverflow.com/questions/12312926/why-use-handler> (дата звернення: 02.11.2024)
4. Відеоурок YouTube Ukrainian android dev «Android УКРАЇНСЬКОЮ Урок 21 Recycler View Простий приклад» URL: <https://www.youtube.com/watch?v=J7NWMgi09MA> (дата звернення: 02.11.2024)
5. Документація AndroidDevelopers Recycler View. URL: <https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView> (дата звернення: 02.11.2024)

ДОДАТОК А

MainActivity.java

package ievsieieva.oleksandra.nure;  
  
import android.app.DatePickerDialog;  
import android.app.Dialog;  
import android.app.ProgressDialog;  
import android.app.TimePickerDialog;  
import android.content.Context;  
import android.content.DialogInterface;  
import android.os.Bundle;  
import android.os.Handler;  
import android.os.Looper;  
import android.util.Log;  
import android.view.View;  
import android.widget.Button;  
import android.widget.DatePicker;  
import android.widget.TimePicker;  
import android.widget.Toast;  
  
import androidx.activity.EdgeToEdge;  
import androidx.appcompat.app.AlertDialog;  
import androidx.appcompat.app.AppCompatActivity;  
import androidx.core.graphics.Insets;  
import androidx.core.view.ViewCompat;  
import androidx.core.view.WindowInsetsCompat;  
import androidx.recyclerview.widget.RecyclerView;  
  
import java.util.ArrayList;  
import java.util.Calendar;  
  
public class MainActivity extends AppCompatActivity {  
 String TAG = MainActivity.class.getSimpleName();  
 Button alert;  
 Button progress;  
 Button date;  
 Button time;  
 Button custom;  
 Context context = this;  
  
  
 @Override  
 protected void onCreate(Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 EdgeToEdge.*enable*(this);  
 setContentView(R.layout.*activity\_main*);  
 ViewCompat.*setOnApplyWindowInsetsListener*(findViewById(R.id.*main*), (v, insets) -> {  
 Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.*systemBars*());  
 v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);  
 return insets;  
 });  
  
 alert = findViewById(R.id.*alert*);  
 progress = findViewById(R.id.*progress*);  
 date = findViewById(R.id.*date*);  
 time = findViewById(R.id.*time*);  
 custom = findViewById(R.id.*custom*);  
  
 alert.setOnClickListener(v ->  
 new AlertDialog.Builder(context)  
 .setTitle("Alert dialog")  
 .setMessage("Alert dialog test")  
 .setPositiveButton("Ok", (dialog, which) -> {  
 Toast.*makeText*(context, "Alert dialog: okay", Toast.*LENGTH\_LONG*).show();  
 })  
 .setNegativeButton("Cancel", ((dialog, which) -> {  
 Toast.*makeText*(context, "Alert dialog: cancel", Toast.*LENGTH\_LONG*).show();  
 }))  
 .create()  
 .show()  
 );  
  
 progress.setOnClickListener( v -> {  
 ProgressDialog dialog = new ProgressDialog(context);  
 dialog.setProgressStyle(ProgressDialog.*STYLE\_SPINNER*);  
 dialog.setMessage("In progress...");  
 dialog.create();  
 dialog.show();  
 }  
 );  
  
 date.setOnClickListener(v -> {  
 final Calendar calendar = Calendar.*getInstance*();  
 new DatePickerDialog(context, (view, year, month, dayOfMonth) -> {  
 Toast.*makeText*(context, String.*format*("year: %d, month: %d, day: %d", year, month, dayOfMonth), Toast.*LENGTH\_LONG*).show();  
 }, calendar.get(Calendar.*YEAR*), calendar.get(Calendar.*MONTH*), calendar.get(Calendar.*DAY\_OF\_MONTH*))  
 .show();  
 }  
 );  
  
 time.setOnClickListener(v -> {  
 final Calendar calendar = Calendar.*getInstance*();  
 new TimePickerDialog(context, (view, hourOfDay, minute) -> {  
 Toast.*makeText*(context, String.*format*("hours: %d, minutes: %d", hourOfDay, minute), Toast.*LENGTH\_LONG*).show();  
 }, calendar.get(Calendar.*HOUR\_OF\_DAY*), calendar.get(Calendar.*MINUTE*), true)  
 .show();  
 }  
 );  
  
 custom.setOnClickListener(v -> {  
 Dialog dialog = new Dialog(context);  
 dialog.setContentView(R.layout.*dialog\_custom*);  
 dialog.setCancelable(true);  
 dialog.findViewById(R.id.*ok*).setOnClickListener(view -> {  
 dialog.dismiss();  
 Toast.*makeText*(context, "Custom dialog: okay", Toast.*LENGTH\_LONG*).show();  
 });  
 dialog.findViewById(R.id.*cancel*).setOnClickListener(view -> {  
 dialog.dismiss();  
 Toast.*makeText*(context, "Custom dialog: cancel", Toast.*LENGTH\_LONG*).show();  
 });  
 dialog.show();  
 }  
 );  
  
 findViewById(R.id.*handler*).setOnClickListener(v -> {  
 new Handler(Looper.*getMainLooper*()).postDelayed(() -> {  
 Toast.*makeText*(context, "2 seconds after", Toast.*LENGTH\_LONG*).show();  
 }, 2000);  
 });  
  
 ArrayList<String> list = new ArrayList<>();  
 for(int i = 0; i < 100; i++) {  
 list.add("ItemAdapter item with id: ");  
 }  
  
 ((RecyclerView) findViewById(R.id.*list*)).setAdapter(new ItemAdapter(list));  
  
  
 }  
}

ДОДАТОК Б

ItemAdapter.java

package ievsieieva.oleksandra.nure;  
  
import android.view.LayoutInflater;  
import android.view.View;  
import android.view.ViewGroup;  
import android.widget.TextView;  
  
import androidx.annotation.NonNull;  
import androidx.recyclerview.widget.RecyclerView;  
  
import java.util.List;  
  
public class ItemAdapter extends RecyclerView.Adapter<ItemAdapter.ItemViewHolder> {  
 private List<String> \_list;  
  
 public ItemAdapter(List<String> list) {  
 this.\_list = list;  
 }  
  
 @NonNull  
 @Override  
 public ItemViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {  
 View v;  
 if (viewType == 1) {  
 v = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*item*, parent, false);  
 } else {  
 v = LayoutInflater.*from*(parent.getContext())  
 .inflate(R.layout.*item\_grey*, parent, false);  
 }  
 return new ItemViewHolder(v);  
 }  
  
 @Override  
 public void onBindViewHolder(@NonNull ItemViewHolder holder, int position) {  
 holder.id.setText(""+position);  
 holder.text.setText(\_list.get(position) + position);  
 }  
  
 @Override  
 public int getItemViewType(int position) {  
 return position%2;  
 }  
  
 @Override  
 public int getItemCount() {  
 return \_list.size();  
 }  
  
 public static class ItemViewHolder extends RecyclerView.ViewHolder {  
 TextView id;  
 TextView text;  
 public ItemViewHolder(@NonNull View itemView) {  
 super(itemView);  
 id = itemView.findViewById(R.id.*id*);  
 text = itemView.findViewById(R.id.*text*);  
  
 }  
 }  
}

ДОДАТОК В

Item.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:paddingStart="8dp"

android:paddingEnd="8dp"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools">

<TextView

android:id="@+id/id"

tools:text="10"

android:layout\_marginTop="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintBottom\_toTopOf="@id/text"

/>

<TextView

android:id="@+id/text"

tools:text="Item Id: 10"

android:layout\_marginBottom="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toBottomOf="@id/id"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"

/>

</androidx.constraintlayout.widget.ConstraintLayout>

Та Item\_grey.xml

<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:background="#9E9E9E"

android:paddingStart="8dp"

android:paddingEnd="8dp"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools">

<TextView

android:id="@+id/id"

tools:text="10"

android:layout\_marginTop="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintBottom\_toTopOf="@id/text"

/>

<TextView

android:id="@+id/text"

tools:text="Item Id: 10"

android:layout\_marginBottom="16dp"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

app:layout\_constraintTop\_toBottomOf="@id/id"

app:layout\_constraintStart\_toStartOf="parent"

app:layout\_constraintEnd\_toEndOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"

/>

</androidx.constraintlayout.widget.ConstraintLayout>