MINISTRY FOR DEVELOPMENT OF INFORMATION TECHNOLOGIES AND COMMUNICATIONS OF THE REPUBLIC OF UZBEKISTAN

TASHKENT UNIVERSITY OF INFORMATION TECHNOLOGIES
NAMED AFTER MUHAMMAD AL-KHWARIZMI

Practical work #2

By subject

"Mobile Application Development"

Completed by: Tulaganov Otabek
Group MAD 401- 1
Received by:Xushbaqov Sherzod

Tashkent 2025

Theoretical part of task

Questions:

1. What is a dialog window, and what are its main types?
2. What methods are available for data transfer between activities?
3. What is the main difference between Property Animator and View Animator?
4. What are the key differences between RecyclerView and ListView?

Answers

1. A **dialog window** (or simply **dialog**) in mobile development is a small interface element that pops up on top of the main app content to prompt the user for a decision, show important information, or request input. It's typically modal, meaning it blocks interaction with the rest of the app until it's dismissed.

Alert Dialog

- Displays a message and optional actions (e.g., OK, Cancel).
- Used for warnings, confirmations, or information.
- Example: “Are you sure you want to delete this item?”

Confirmation Dialog

- A specific type of alert that asks the user to confirm or reject an action.
- Usually includes two buttons: Confirm / Cancel.

Prompt/Input Dialog

- Requests user input (like entering text or selecting a value).
- Example: A dialog asking for a username.

Progress Dialog (Loading Spinner)

- Shows ongoing background operations (e.g., file upload or data fetching).
- Can be indeterminate (e.g., spinner) or determinate (shows progress bar).

Bottom Sheet Dialog (Android specific but also seen in cross-platform)

- Slides up from the bottom.
- Can show menus, actions, or forms.
- Comes in **modal** (blocks interaction) or **persistent** (still allows some interaction with the main screen).

Custom Dialog

- Developers can create custom UIs inside dialog windows to match the app's design or needs.
- Useful for more complex interactions (e.g., forms, media previews).

2. Here's how you can pass data between **activities in Android** , with different methods:

---

1. Using Intent Extras (Simple data)

Sender Activity:

```
Intent intent = new Intent(CurrentActivity.this,
SecondActivity.class);
intent.putExtra("username", "JohnDoe");
startActivity(intent);
```

Receiver Activity (SecondActivity):

```
String username = getIntent().getStringExtra("username");
```

---

2. Using Bundle

Sender:

```
Intent intent = new Intent(CurrentActivity.this,
SecondActivity.class);
Bundle bundle = new Bundle();
bundle.putString("email", "john@example.com");
bundle.putInt("userId", 123);
intent.putExtras(bundle);
startActivity(intent);
```

Receiver:

```
Bundle extras = getIntent().getExtras();
if (extras != null) {
    String email = extras.getString("email");
    int userId = extras.getInt("userId");
}
```

---

3. Passing Serializable Objects

Create a Serializable class:

```
public class User implements Serializable {
    String name;
    int age;

    public User(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Sender:

```
User user = new User("Alice", 25);
Intent intent = new Intent(CurrentActivity.this,
SecondActivity.class);
intent.putExtra("user", user);
startActivity(intent);
```

Receiver:

```
User user = (User)
getIntent().getSerializableExtra("user");
```

---

4. Using Static Variables (not ideal for long-term use)

Create a static data holder:

```
public class DataHolder {
    public static User user;
}
```

Sender:

```
DataHolder.user = new User("Bob", 30);
Intent intent = new Intent(CurrentActivity.this,
SecondActivity.class);
startActivity(intent);
```

Receiver:

```
User user = DataHolder.user;
```

---

3. The **main difference between Property Animator and View Animator** in mobile development (specifically Android) lies in **how** they perform animations and **what** they can animate.

View Animator (Legacy - Pre-Honeycomb)

- Uses **View animation** (like AlphaAnimation, TranslateAnimation, etc.).
- **Only animates the visual appearance** of views — the actual property values (like X, Y, rotation) **don't change**.
- The view returns to its original state after the animation ends.
- Suitable for **simple 2D animations** (fade, scale, move).

Example:

```
TranslateAnimation anim = new TranslateAnimation(0, 100,
0, 0);
anim.setDuration(1000);
myView.startAnimation(anim);
```

Even after moving, `myView.getX()` still returns the **original position** — only the visual is animated.

Property Animator (Modern - Introduced in Android 3.0+)

- Uses `ObjectAnimator` and `AnimatorSet` under the **Property Animation** system.
- **Actually changes property values** of views — like `alpha`, `translationX`, `rotation`, etc.
- More flexible: supports complex animations, chaining, custom properties, and interpolators.
- Preferred for modern apps.

Example:

```
ObjectAnimator animator = ObjectAnimator.ofFloat(myView,
"translationX", 0f, 100f);
animator.setDuration(1000);
animator.start();
```

✅After this runs, `myView.getTranslationX()` will return `100f`.

Summary Table

Feature	View Animator	Property Animator
Affects actual properties?	❌No	✅Yes
Back to original state after animation?	✅Yes	❌No
API Level introduced	API 1	API 11 (Honeycomb)
Suitable for	Simple visual effects	Complex and interactive animations
Example classes	TranslateAnimation, AlphaAnimation	ObjectAnimator, AnimatorSet

4. Both are used to display scrollable lists of items, but **RecyclerView** is the modern, flexible, and powerful replacement for **ListView**.

---

1. ViewHolder Pattern

Feature	ListView	RecyclerView
ViewHolder pattern	Optional (manual)	Required and built-in

- RecyclerView **forces** use of the ViewHolder pattern to improve performance and reduce unnecessary findViewById calls.
- ListView allows it but doesn't enforce it.

---

2. Layout Flexibility

Feature	ListView	RecyclerView
Layout types	Only vertical list	Vertical, horizontal, grid, staggered, custom (via LayoutManager)

- RecyclerView is far more flexible thanks to LayoutManagers like:
- LinearLayoutManager (vertical/horizontal)
- GridLayoutManager
- StaggeredGridLayoutManager

---

3. Animations and Item Effects

Feature	ListView	RecyclerView
Animations	Manual	Built-in support for animations (add, remove, move, change)

- RecyclerView supports item animations natively.
- In ListView, you'd have to implement custom animations.

---

4. Performance

Feature	ListView	RecyclerView
Performance	OK	Better, especially with large datasets

- RecyclerView handles large and dynamic data sets more efficiently.

---

5. Customization

Feature	ListView	RecyclerView
Decorators, snapping, swiping, drag & drop	Hard to implement	Easy using ItemDecoration, ItemTouchHelper, etc.

RecyclerView supports:

Custom dividers (ItemDecoration)

Swiping items to delete

Drag and drop

Paging, snapping, etc.

---

6. Adapter

Feature	ListView	RecyclerView
Adapter	ArrayAdapter / BaseAdapter	Must create a custom RecyclerView.Adapter with ViewHolder

RecyclerView adapters are more complex but also more powerful and flexible.

✅**Summary Table**

Feature	ListView	RecyclerView
ViewHolder Pattern	Optional	Required
Layout Types	Single vertical	Multiple (vertical, horizontal, grid, etc.)
Performance	Lower	Higher
Animation Support	Minimal	Built-in
Customization	Limited	Extensive
Adapter Simplicity	Simpler	More flexible but more complex
Future Usage	Deprecated for most cases	Recommended

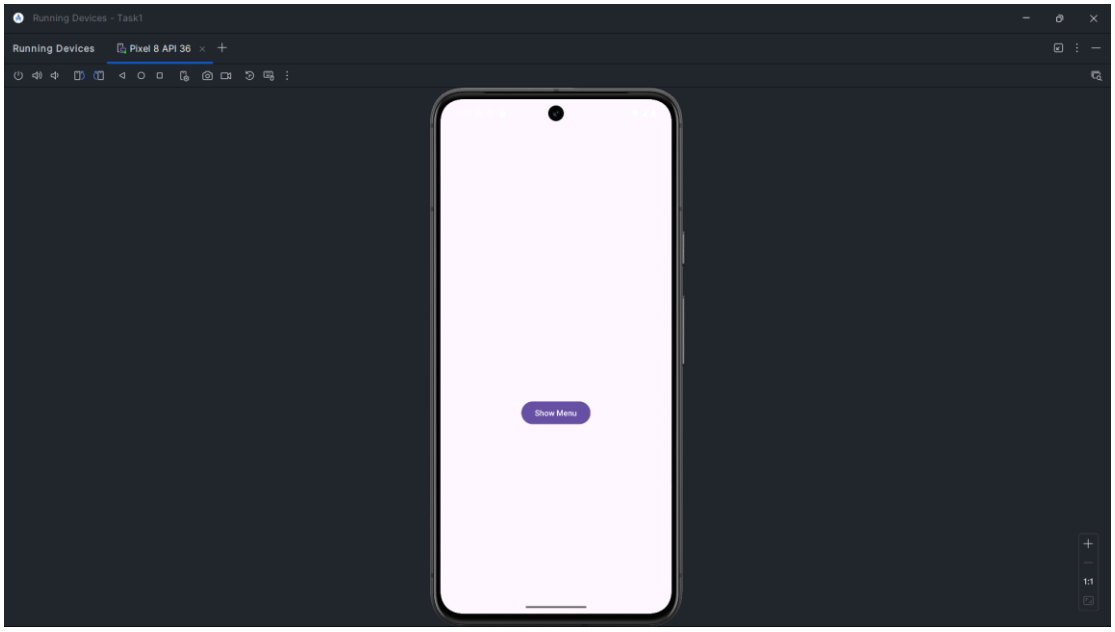
If you’re building a modern Android app, **RecyclerView is the go-to choice**.

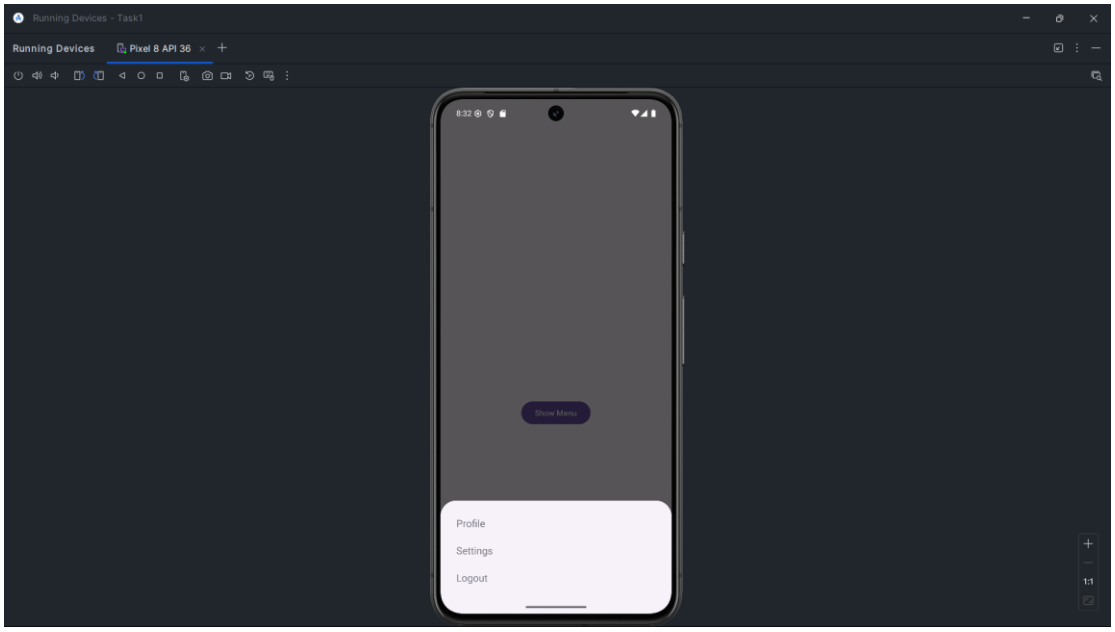
Practical part of task

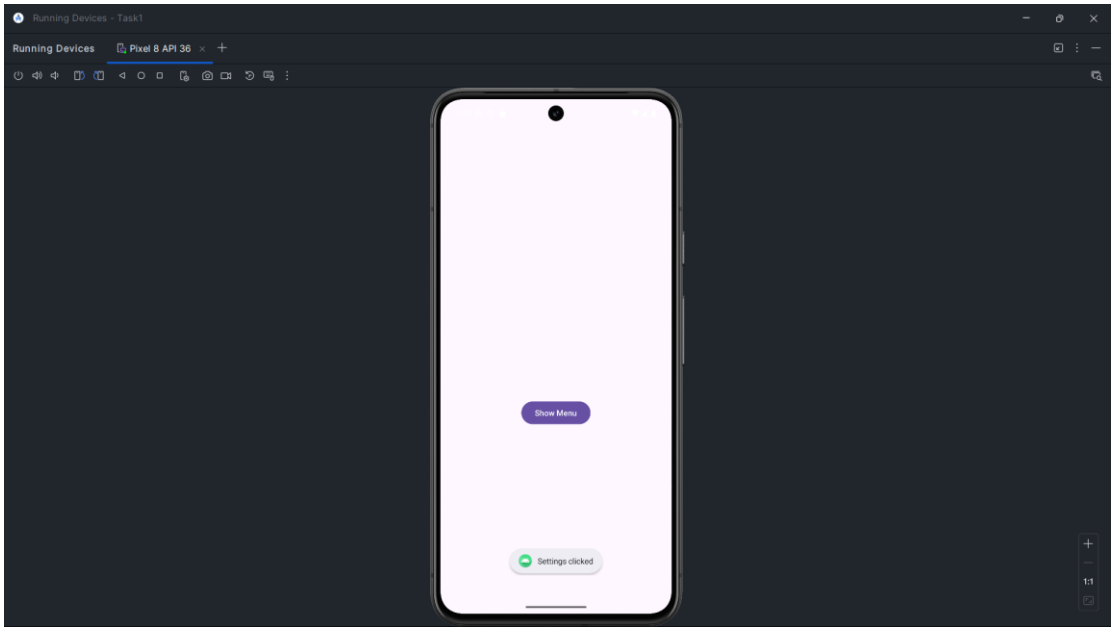
1. **Adding BottomSheetDialog**

• Create a bottom sheet menu (BottomSheetDialog) that appears when a button is clicked.

• The menu should include options such as "Profile," "Settings," and "Logout."


Running Devices - Task1
Running Devices
Pixel 8 API 36
Show Menu
1:1


Running Devices - Task1
Running Devices
Pixel 8 API 36
8:32
Show Menu
Profile
Settings
Logout
1:1

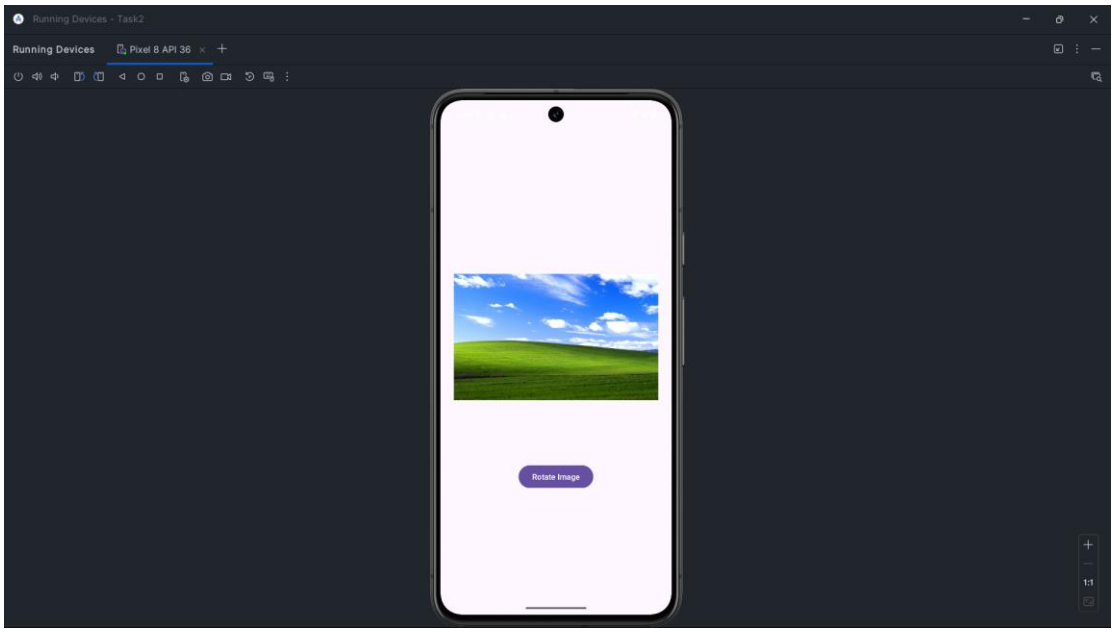


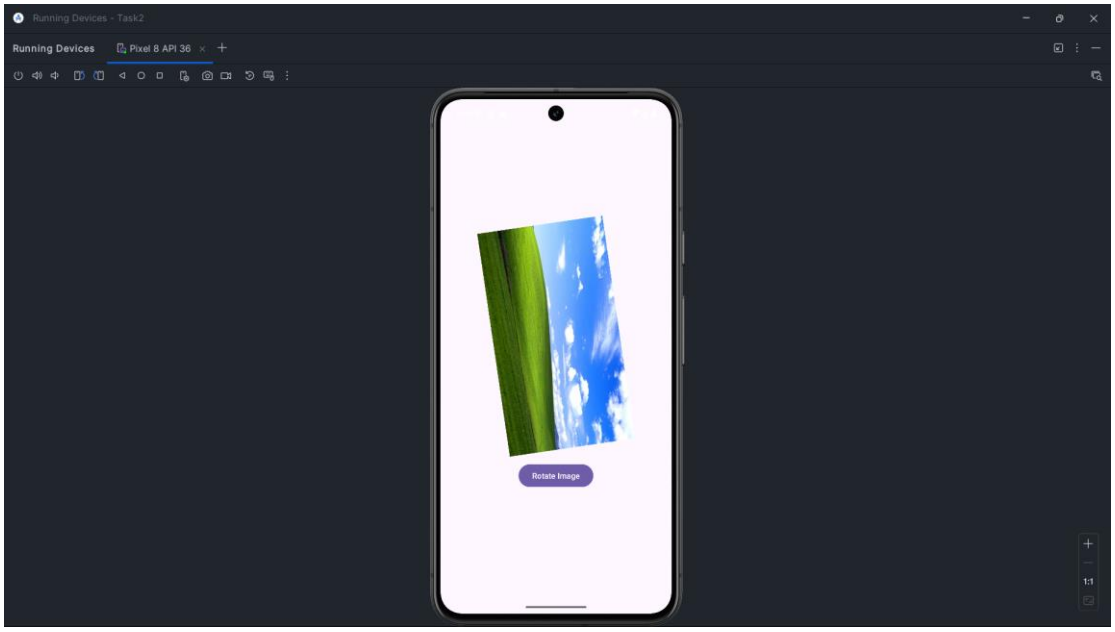
Source code below:

https://github.com/MrTolaganov/android-development/tree/main/task1

2. **Rotating an Image with ObjectAnimator**

• Ensure that an image rotates 360 degrees when a button is clicked.



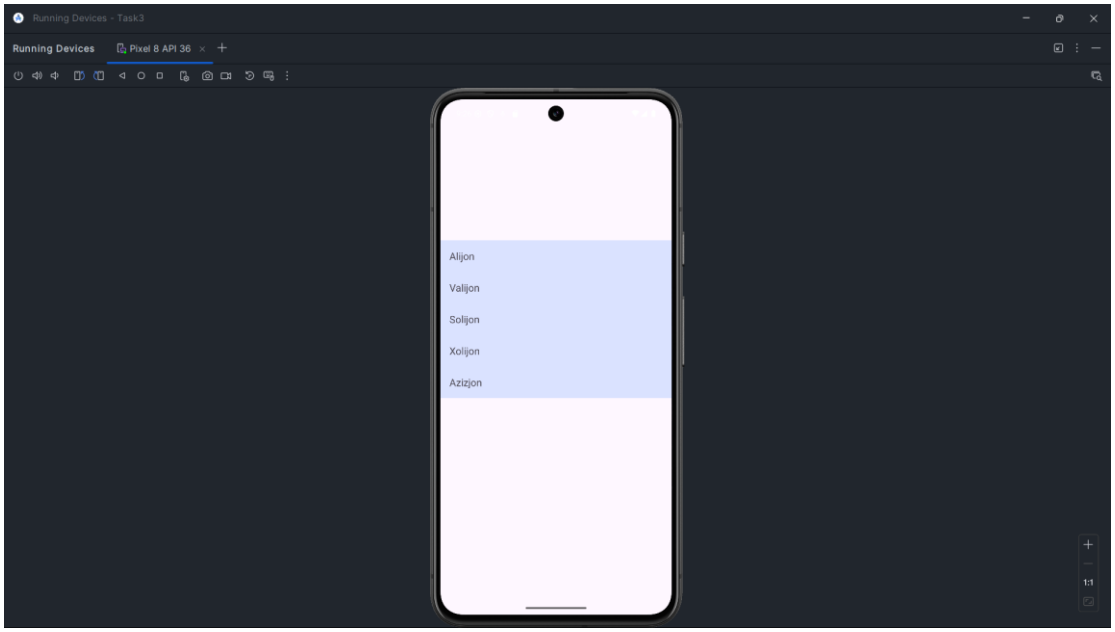


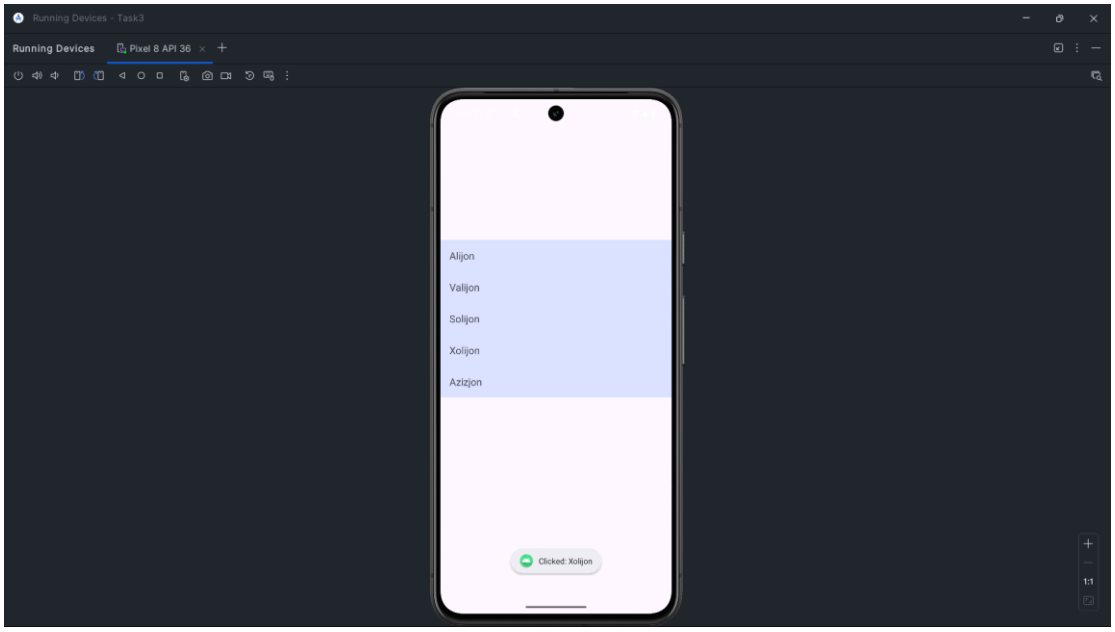
Source code below:

https://github.com/MrTolaganov/android-development/tree/main/task2

3. **Adding ClickListener to RecyclerView**

• Implement a ClickListener for RecyclerView items so that clicking an item displays its name using a Toast message.





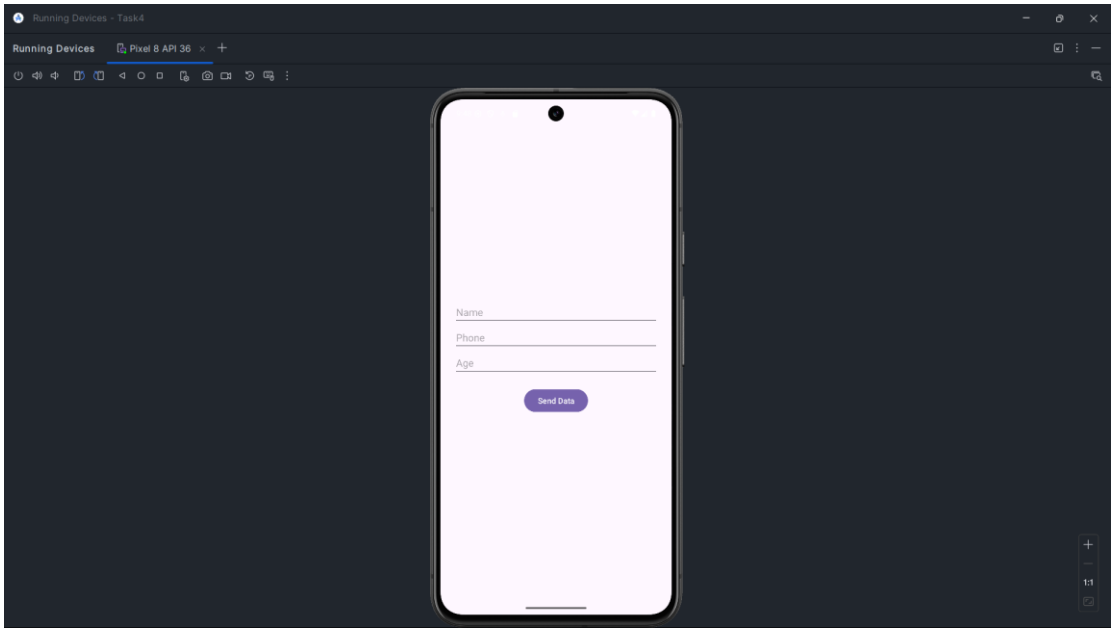
Source code below:

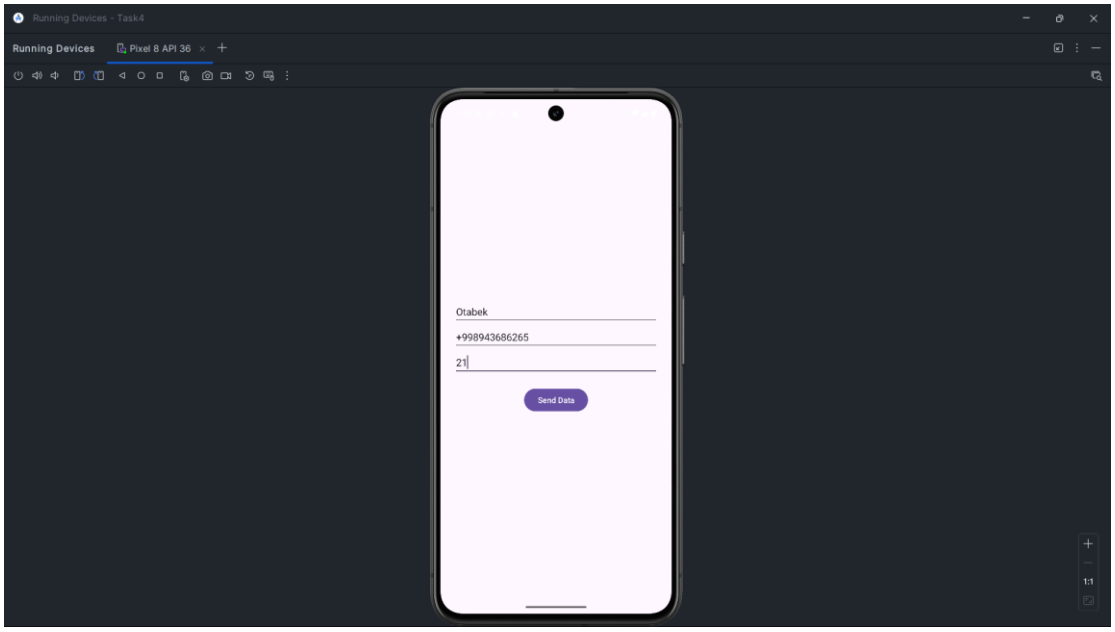
https://github.com/MrTolaganov/android-development/tree/main/task3

4. **Using a Bundle Object**

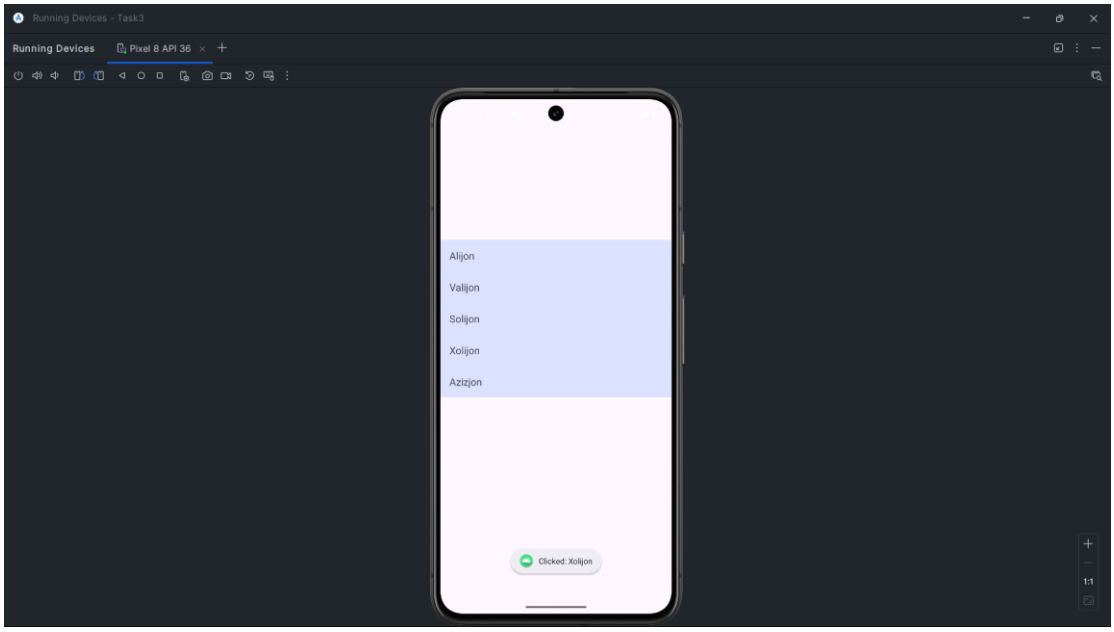
• In Activity A, the user enters their phone number, name, and age.

• Pass this data to Activity B using a Bundle and display it on the screen.




Running Devices - Task4
Running Devices
Pixel 8 API 36
Otabek
+998943686265
21
Send Data
1:1


Running Devices - Task4
Running Devices
Pixel 8 API 36
Name: Otabek
Phone: +998943686265
Age: 21
1:1



Source code below:

https://github.com/MrTolaganov/android-development/tree/main/task4

Conclusion

In this set of mobile development tasks, we explored fundamental techniques for building interactive Android apps using Java. First, we learned how to use a Bundle to pass user input such as name, phone number, and age from one activity to another, allowing seamless data sharing between screens. Next, we ensured both activity layouts (activity_a.xml and activity_b.xml) display content vertically centered on the screen for a cleaner, more user-friendly interface. We also removed the default MainActivity and updated the AndroidManifest.xml to make ActivityA the new launcher activity. Lastly, we implemented a click listener for a RecyclerView so that when a user taps an item, its name is displayed using a Toast message, enhancing interactivity. These

foundational skills help create smooth navigation, data flow, and user experiences in Android apps.