Практична робота 6. Проектування графічного інтерфейсу користувача в Android Studio

Теоретичні відомості

При створенні екранів графічного інтерфейсу користувача успадковується клас Activity і використовуються View для взаємодії з користувачем.

Кожна активність - це екран (за аналогією з формою), який додаток може показувати користувачам. Чим складніший додаток, тим більше екранів (активностей) буде потрібно. При створенні додатку потрібно, як мінімум, початковий (головний) екран, який забезпечує основу користувальницького інтерфейсу. При необхідності цей інтерфейс доповнюється другорядними активностями, що призначені для введення інформації, її виведення та надання додаткових можливостей. Запуск (або повернення з) нової активності призводить до «переміщення» між екранами користувацького інтерфейсу.

Більшість активностей проектуються таким чином, щоб використовувати весь екранний простір, але можна також створювати напівпрозорі або плаваючі діалогові вікна.

Життєвий цикл активності

Для створення нової активності успадковується клас Activity. Всередині реалізації класу необхідно визначити користувальницький інтерфейс і реалізувати необхідний функціонал. Базовий каркас для нової активності показаний нижче:

```
package com.example.myapplication;
import android.app.Activity;
import android.os.Bundle;
public class MyActivity extends Activity {
    /** Викликається при створенні Активності */
```

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

Базовий клас Activity являє собою порожній екран, тому перше, що потрібно зробити, це створити користувальницький інтерфейс за допомогою View і розмітки (Layout).

View - це елементи UI, які відображають інформацію і забезпечують взаємодію з користувачем. Android надає кілька класів розмітки (Layout), що також називаються View Groups, які можуть містити всередині себе кілька View, для створення користувацького інтерфейсу програми.

Щоб призначити користувальницький інтерфейс для активності, всередині обробника onCreate використовується метод setContentView:

```
@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TextView textView = new TextView(this);
    setContentView(textView);
}
```

У цьому прикладі як UI для активності виступає об'єкт класу TextView.

При створенні реальних додатків частіше застосовується метод проектування, що використовує ресурси програми, відокремлені від коду. Такий підхід дозволяє створювати додатки, що забезпечують високу якість реалізації UI, не залежно від умов роботи програми: додатки пропонують зручний для користувача мовний інтерфейс (залежить від локалізації), слабо залежать від вирішення і розмірів екрану і т. д.). Найголовніше, що така адаптація додатків до нових умов не вимагає змін в коді програми, потрібно тільки забезпечити необхідні ресурси (картинки, локалізовані рядки і т.д.). Стандартний для Android підхід показаний нижче:

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

```
        setContentView(R.layout.main);
    }
```

Для використання активності в додатку її необхідно зареєструвати в файлі маніфесті шляхом додавання елемента <activity> всередині вузла <application>, в іншому випадку її неможливо буде використовувати. Нижче показано, як створити елемент <activity> для активності MyActivity:

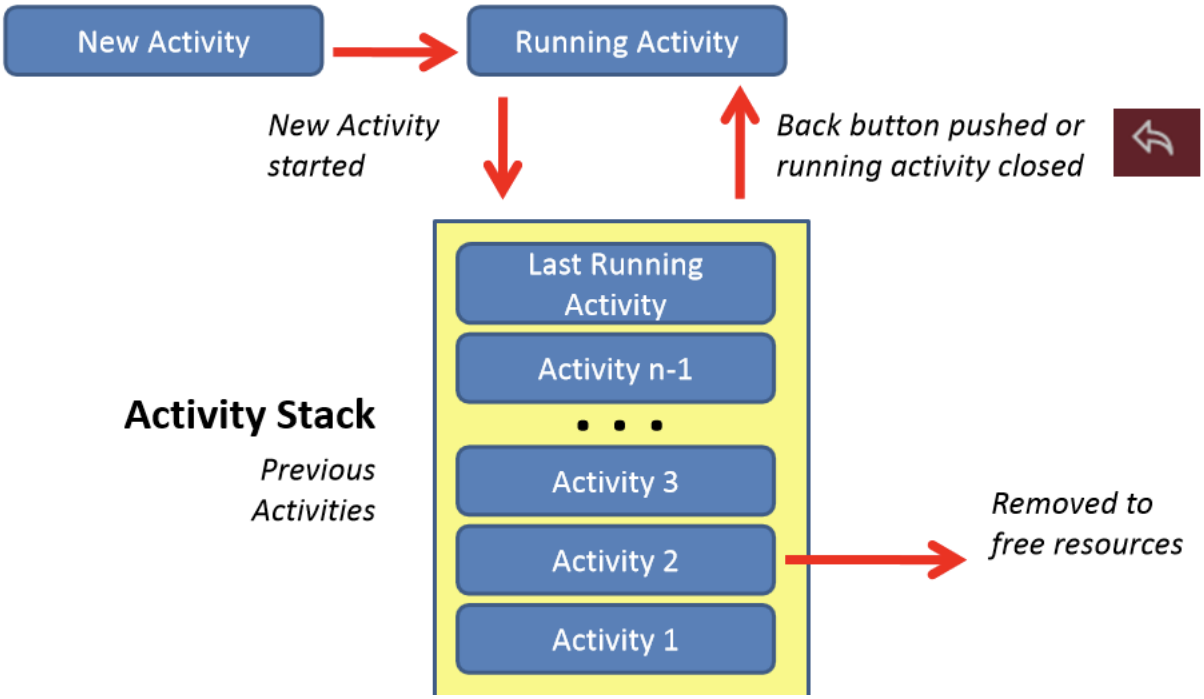
```
<activity android:label="@string/app_name"
android:name=".MyActivity">
</activity>
```

У тезі <activity> можна додавати елементи <intent-filter> для вказівки Намірів (Intent), які активність буде відстежувати. Кожен «фільтр намірів» визначає одну або кілька дій (action) і категорій (category), які підтримуються активністю. Важливо знати, що активність буде доступна з головного меню запуску додатків тільки у випадку, якщо в маніфесті для неї вказано <intent-filter> для дії MAIN і категорії LAUNCHER, як показано у прикладі:

```
<activity android:label="@string/app_name"
android:name=".MyActivity">
<intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Для створення додатків, які будуть правильно керувати ресурсами, що надають користувачеві зручний інтерфейс, важливо добре розуміти життєвий цикл активності. Це пов'язано з тим, що додатки Android не можуть контролювати свій життєвий цикл, ОС сама керує всіма процесами, і як наслідок активностями в середині них. При цьому, стан активності допомагає ОС визначити пріоритет батьківського додатку для цієї активності (Application). А пріоритет додатку впливає на те, з якою ймовірності його робота (і робота дочірніх активностей) буде перервана системою.

Стан кожної активності визначається її позицією в стеку (LIFO) активностей, запущених в даний момент. При запуску нової активності представлений нею екран розміщується на вершину стека. Якщо користувач натискає кнопку «назад» або ця активність закривається іншим чином, на вершину стека переміщається (і стає активною) активність, що знаходились нижче. Даний процес показаний на поданому нижче рисунку.



На пріоритет додатку впливає його пріоритетна активність. Коли диспетчер пам'яті ОС вирішує, яку програму закрити для звільнення ресурсів, він враховує інформацію про стан активності в стеку для визначення пріоритету додатку.

Стан активностей та відстеження його змін

Активності можуть знаходитися в одному з чотирьох станів:

• Активна (Active). Активність знаходиться на передньому плані (на вершині стека) і має можливість взаємодіяти з користувачем. Android намагатиметься зберегти її працездатність, при необхідності перериваючи роботу інших активних, що знаходяться на нижчих позиціях в стеку для надання

необхідних ресурсів. При виході на передній план іншої активності робота даної активності буде припинена або зупинена.

• Призупинена (Paused). Активність може бути видима на екрані, але не може взаємодіяти з користувачем: в цей момент вона призупинена. Це трапляється, коли на передньому плані знаходяться напівпрозорі або плаваючі (наприклад, діалогові) вікна. Робота призупиненої активності може бути припинена, якщо ОС необхідно виділити ресурси активності переднього плану. Якщо активність повністю зникає з екрану, вона зупиняється.

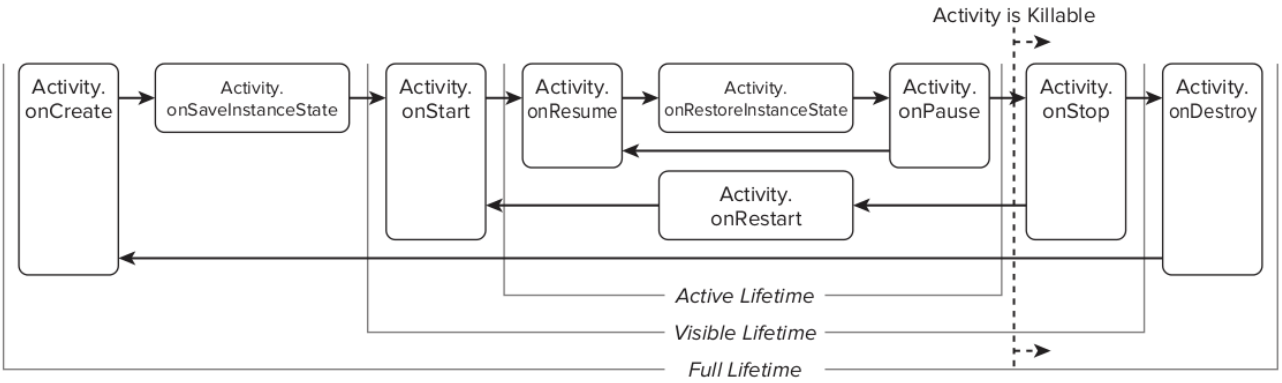
• Зупинена (Stopped). Активність невидима, вона знаходиться в пам'яті, зберігаючи інформацію про свій стан. Така активність стає кандидатом на передчасне закриття, якщо системі буде потрібно пам'ять для чогось іншого. При зупинці Активності розробнику важливо зберегти дані і поточний стан користувача інтерфейсу (стан полів введення, позицію курсора і т.д.). Якщо активність завершує свою роботу або закривається, вона стає неактивною.

• Неактивна (Inactive). Коли робота активності завершена, і перед тим, як вона буде запущена, дана активність знаходиться в неактивному стані. Такі активності видаляються з стека і повинні бути (пере) запущені, щоб їх можна було використовувати.

Зміна стану додатка - недетермінований процес, що керується виключно менеджером пам'яті Android. При необхідності Android спочатку закриває додатки, що містять неактивні активності, потім зупинені і, в крайньому випадку, припинені.

Для забезпечення повноцінного інтерфейсу додатку, зміни його стану повинні бути непомітні для користувача. Міняючи свій стан з призупиненого на зупинений або з неактивного на активний, активність не повинна зовні змінюватися. При зупинці або призупиненні роботи активності розробник повинен забезпечити збереження стану активності, щоб її можна було відновити при виході активності на передній план. Для цього в класі Activity міститься обробники подій, перевизначення яких дозволяє розробнику відслідковувати зміну станів активності.

Обробники подій класу Activity дозволяють відслідковувати зміни станів відповідного об'єкта Activity під час усього життєвого циклу, що відображає рисунок поданий нижче.



Показаний приклад із заглушками для таких методів - обробників подій:

package com.example.myapplication; import android.app.Activity;

import android.os.Bundle;

public class MyActivity extends Activity {

// Викликається при створені Активності @Override

public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);

// Ініціалізує Активність.

}

// Викликається після завершення метода onCreate

// Використовується для відновлення стану UI @Override

public void onRestoreInstanceState(Bundle savedInstanceState) { super.onRestoreInstanceState(savedInstanceState);

// Відновити стан UI з об’єкта savedInstanceState.

// Даний об’єкт також був переданий методу onCreate.

}

// Викликається перед тим, як Активність знову стає видимою @Override

public void onRestart(){ super.onRestart();

// Відновити стан UI з урахуванням того,

// що дана Активність вже була видима.

}

```
// Викликається коли Активність стала видима @Override
public void onStart(){ super.onStart();
// Проробити необхідні дії для
// Активності, видимої на экрані
}
// Повинен викликатись на початку видимого стану.
// Насправді Android викликає даний обробник тільки
// для Активностей, відновлених з неактивного стану @Override
public void onResume(){ super.onResume();
// Відновити призупинені оновлення UI,
// потоки и процеси, «заморожені», коли
// Активність була в неактивному стані
}
// Викликається перед виходом з активного стану,
// дозволяючи зберегти стан в об’єкті savedInstanceState @Override
public void onSaveInstanceState(Bundle savedInstanceState) {
// Об’єкт savedInstanceState буде в подальшому
// переданий методам onCreate і onRestoreInstanceState
super.onSaveInstanceState(savedInstanceState);
}
// Викликається перед виходом з активного стану @Override
public void onPause(){
// «Заморозити» оновлення UI, потоки чи
// «трудомісткі» процеси, непотрібні, коли Активність
// не на передньому плані super.onPause();
}
// Викликається перед виходом з видимого стану @Override
public void onStop(){
// «Заморозити» оновлення UI, потоки чи
// «трудомісткі» процеси, непотрібні, коли Активність
```

```
// не на передньому плані.
// Зберегти всі дані і зміни в UI, так як
// процес може бути в будь-який момент вбитий системою super.onStop();
}
// Викликається перед знищенням активності @Override
public void onDestroy(){
// Звільнити всі ресурси, включаючи працюючі потоки,
// з’єднання з БД і т. д. super.onDestroy(); } }
```

Робота із ресурсами

Незалежно від використовуваного середовища розробки, досить розумно відокремлювати використовувані додатком ресурси від коду. Зовнішні ресурси легше підтримувати, оновлювати і контролювати. Така практика також дозволяє описувати альтернативні ресурси для підтримки додатком різних пристроїв та реалізовувати локалізацію додатків.

Додатки Android використовують різноманітні ресурси із зовнішніх (по відношенню до коду) файлів, від простих (рядки і кольору) до більш складних (зображення, анімації, візуальні стилі). Надзвичайно корисно також відокремлювати від коду такі важливі ресурси, як розмітки екранів (Layout), використовувані в активності. Android автоматично вибирає найбільш підходящі варіанти з дерева ресурсів додатків, що містять різні значення для різних апаратних конфігурацій, мов і регіонів, не вимагаючи при цьому жодного рядка коду.

Ресурси програми зберігаються в каталозі res в дереві каталогів проекту. Плагін ADT автоматично створює каталог res з підкаталогами values, layout і drawable- *, в яких зберігаються, відповідно: строкові константи, розмітка за замовчуванням іконка програми.

Для дев'яти головних типів ресурсів використовуються різні підкаталоги каталогу res, це:

– прості значення,

– зображення,
– розмітка,
– анімація,
– стилі,
– меню,
– налаштування пошуку,
– XML,
– «сирі» дані.

При збірці пакету .apk ці ресурси максимально ефективно компілюються і включаються в пакет.

Для роботи з ресурсами всередині коду плагін ADT автоматично генерує файл класу R, що містить посилання на всі ресурси. Імена файлів ресурсів можуть містити тільки латинські букви в нижньому регістрі, підкреслення і крапки.

Android підтримує наступні типи значень: рядки, кольори, розміри і масиви (рядкові та цілочисельні). Ці дані зберігаються у вигляді XML-файла в каталозі res/values. Нижче показано приклад подібного файлу:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">To Do List</string>
<color name="app_background">#FF0000FF</color>
<dimen name="default_border">5px</dimen>
<array name="string_array">
<item>Item 1</item>
<item>Item 2</item>
<item>Item 3</item>
</array>
<array name="integer_array">
<item>3</item>
<item>2</item>
```

<item>1</item>

</array>

</resources>

У прикладі містяться всі доступні типи простих значень, але насправді кожен тип ресурсів зберігається в окремому файлі, наприклад, res/values/arrays.xml містить масиви, а res/values/strings.xml - строкові константи.

Строкові константи визначаються за допомогою тега <string>, як показано на прикладі:

<string name = "greeting_msg"> Привіт! </ string>

Для виділення тексту в рядках можна використовувати HTML-теги

<b>, <i> і <u>.

приклад:

<string name = "greeting_msg"> <b> Привіт </ b>, дарлінг! </ string>

При необхідності використання даного рядка в коді програми використовується вищезгаданий клас R:

String greeting = getString (R.string.greeting_msg);

Робота із кольорами, зображеннями, стилями та темами

Для опису кольорів використовується тег <color>. Значення кольору вказуються в шістнадцятковому вигляді в одному з наступних форматів:

- # RGB
- # ARGB
- # RRGGBB
- # AARRGGBB

У прикладі показано опис напівпрозорого червоного кольору і непрозорого зеленого:

<color name="transparent_red">#77FF0000</color>

<color name="opaque_green">#0F0</color>

Посилання на розміри найчастіше зустрічаються всередині ресурсів зі стилями і розміткою, наприклад, при вказівці товщини рамки або величини

шрифту. Для опису розмірів використовується тег <dimen> із зазначенням виду розмірності:

- px - реальні екранні пікселі,
- in - фізичні дюйми,
- pt - 1/72 дюйма, обчислюється з фізичного розміру екрану,
- mm - фізичні міліметри, обчислюється з фізичного розміру екрану,
- dp - «незалежні» від щільності екрану пікселі, дорівнюють одному пікселю при еталонній щільності 160 dpi; можна також вказувати як dip; найчастіше використовуються для вказання розмірів рамок і полів,
- sp - «незалежні» від масштабу пікселі, аналогічні dp, але враховують також налаштування користувацького шрифту (великий, дрібний, середній), тому рекомендуються для опису шрифтів.

Приклад опису «великого» шрифту і «стандартної» рамки:

<dimen name = "standard_border"> 5dp </ dimen>

<dimen name = "large_font_size"> 16sp </ dimen>

Стилі і теми дозволяють підтримувати єдність зовнішнього вигляду програми за допомогою атрибутів, використовуваних View, найчастіше це кольори і шрифти. Зовнішній вигляд програми легко змінюється при зміні стилів (тем оформлення) в маніфесті додатків.

Для створення стилю використовується тег <style> з атрибутом name, що містить елементи <item>, кожен з яких, у свою чергу, також має атрибут name, який вказує тип параметра (наприклад, колір або розмір). Всередині елемента <item> зберігається значення параметра:

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

<resources>

<style name = "StyleName">

<item name = "attributeName"> attributeValue </ item> [... Ще елементи <item> ...]

</ style>

</ resources>

У тегу <style> можна вказати атрибут parent, що робить можливим

«успадкування» стилів при необхідності внести в новий стиль незначні відмінності від наявного:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="NormalText">
<item name="android:textSize">14sp</item>
<item name="android:textColor">#111</item>
</style>
<style name="SmallText" parent="NormalText">
<item name="android:textSize">8sp</item>
</style>
</resources>
```

Ресурси Drawable містять растрові зображення. Це можуть бути складні складові ресурси, такі, як LevelListDrawables і StateListDrawables, подані у форматі XML, а також растрові зображення NinePatch. Ресурси Drawable зберігаються в каталогах res / drawable - у вигляді окремих файлів. Ідентифікаторами для таких ресурсів служать імена в нижньому регістрі (без розширення). Підтримуються формати PNG (рекомендований), JPEG та GIF.

Завдяки використанню ресурсів з розміткою (layout) розробник має можливість відокремити логіку програми від її зовнішнього вигляду. Розмітку, визначену у файлі формату XML, можна завантажити для використання в активності методом setContentView. Це реалізовано в поданому прикладі у методі onCreate:

```
setContentView (R.layout.main);
```

Кожен ресурс з розміткою зберігається в окремому файлі в каталозі res

/ layout. Файл використовується як ідентифікатор даного ресурсу (як звичайно, без розширення). При створенні навчального додатку у попередній роботі майстер створення нових проектів створив файл res/layout/main.xml, який піддавався редагуванню:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="fill_parent" android:layout_height="fill_parent" android:background="@color/screen_bkg_color" android:orientation="vertical" >

<TextView

android:layout_width="fill_parent" android:layout_height="wrap_content" android:background="@color/view_bkg_color" android:text="@string/hello" android:textColor="@color/text_color" />

</LinearLayout>

Даний файл містить розмітку LinearLayout, яка є контейнером для елемента TextView, відображає вміст рядка з ім'ям hello, описану в ресурсі strings.

Анімація в Android

Android підтримує два види анімації: покрокову анімацію, послідовно виводить на екран зображення з заданою тривалістю, і анімацію, що базується на розрахунку проміжних кадрів, в цьому випадку застосовуються різні перетворення - обертання, розтягування, переміщення і затемнення. Всі ці трансформації описуються в XML-файлі в каталозі res / anim. Приклад файлу анімації, в якому цільовий елемент одночасно повертається на 270 градусів, стискається і поступово зникає:

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

<set xmlns:android="http://schemas.android.com/apk/res/android" android:interpolator="@android:anim/accelerate_interpolator">

<rotate

android:fromDegrees="0" android:toDegrees="270" android:pivotX="50%" android:pivotY="50%" android:startOffset="500" android:duration="1000" />

<scale

android:fromXScale="1.0" android:toXScale="0.0" android:fromYScale="1.0" android:toYScale="0.0" android:pivotX="50%" android:pivotY="50%" android:startOffset="500" android:duration="500" />

<alpha

android:fromAlpha="1.0" android:toAlpha="0.0" android:startOffset="500" android:duration="500" />

</set>

Ресурс, що описує покрокову анімацію, зберігається в каталозі res / drawable. У наступному прикладі описана анімація, що базується на послідовному відображенні шести зображень поїзда, кожне з яких (крім останнього) відображається протягом 200 мілісекунд. Зрозуміло, для використання такої анімації потрібні ресурси із зображеннями (Drawable), що знаходяться в цьому ж каталозі з іменами (як варіант, у форматі PNG) train1.png .. train6.png.

<animation-list xmlns:android="http://schemas.android.com/apk/res/android" android:oneshot="false">

<item android:drawable="@drawable/train1" android:duration="200" />

<item android:drawable="@drawable/train2" android:duration="200" />

<item android:drawable="@drawable/train3" android:duration="200" />

<item android:drawable="@drawable/train4" android:duration="200" />

<item android:drawable="@drawable/train5" android:duration="200" />

<item android:drawable="@drawable/train6" android:duration="1500" />

</animation-list>

Детальну інформацію про можливості анімації в Android можна знайти тут: http://developer.android.com/guide/topics/graphics/animation.html

Ресурси меню та зовнішні ресурси

Ресурси меню можуть використовуватися для опису як головного меню активності, так і контекстного, що з'являється при тривалому натисканні на який-небудь елемент користувацького інтерфейсу. Меню, описане у форматі XML,

завантажується в додаток з допомогою методу inflate системного сервісу MenuInflater. Зазвичай це відбувається всередині методу onCreateOptionsMenu (для головного меню) або onCreateContextMenu (для контекстного меню), перевизначених в активності. Кожен екземпляр меню описується в окремому файлі XML в каталозі res / menu. Зазвичай імена файлів (без розширень) стають іменами ресурсів. Нижче наведено приклад простого ресурсу з меню, що має три пункти: Refresh, Settings і Quit:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
<item android:id="@+id/menu_refresh" android:title="Refresh" />
<item android:id="@+id/menu_settings" android:title="Settings" />
<item android:id="@+id/menu_quit" android:title="Quit" />
</menu>
```

Кожен з цих пунктів меню має унікальний ідентифікатор (menu_refresh, menu_settings і menu_quit), що дозволяє надалі оброблювачу меню визначити, який з пунктів був обраний користувачем.

Доступ до ресурсів в коді здійснюється за допомогою автоматично згенерованого класу R, точніше, його підкласів. Наприклад, клас R в нашому проекті виглядає так:

```
package com.example.helloandroidworld; public final class R {
public static final class attr {
}
public static final class color {
public static final int screen_bkg_color=0x7f040001; public static final int text_color=0x7f040002;
public static final int view_bkg_color=0x7f040000;
}
public static final class drawable {
public static final int ic_launcher=0x7f020000;
}
```

public static final class layout {

public static final int main=0x7f030000;

}

public static final class string {

public static final int app_name=0x7f050001; public static final int hello=0x7f050000; }}

Члени класів з іменами, відповідними ресурсам, є ідентифікаторами в таблиці ресурсів, а не самими екземплярами ресурсів.

Деякі методи і конструктори можуть брати в якості параметрів ідентифікатори ресурсів, в цьому випадку їх можна використовувати безпосередньо:

setContentView(R.layout.main);
Toast.makeText(this,R.string.awesome_error,Toast.LENGTH_LONG).show()

У випадку, якщо необхідний екземпляр ресурсу, потрібен доступ до таблиці ресурсів, здійснюється за допомогою екземпляра класу Resources. Цей клас містить геттери для всіх видів ресурсів, при цьому в якості параметрів використовуються ідентифікатори ресурсів з класу R:

// Отримуємо доступ до таблиці ресурсів Resources r = getResources();

// і отримуємо необхідні экземпляри ресурсів

CharSequence greetingMsg = r.getText(R.string.greeting_message); Drawable icon = r.getDrawable(R.drawable.app_icon);

int opaqueBlue = r.getColor(R.color.opaque_blue);

float borderWidth = r.getDimension(R.dimen.standard_border); String[] stringArray = r.getStringArray(R.array.string_array); int[] intArray = r.getIntArray(R.array.integer_array);

Для звернення до одного ресурсу всередині опису іншого ресурсу використовується наступна нотація:

attribute="@[packagename:]resourcetype/resourceidentifier"

Ім'я пакету використовується тільки при зверненні до ресурсів з іншого пакета, для звернення до своїх ресурсів його вказувати не потрібно. У нашому випадку таку нотацію можна побачити, наприклад, при описі елемента розмітки TextView у файлі res / layout / main.xml:

<TextView

android:layout_width="fill_parent" android:layout_height="wrap_content" android:background="@color/view_bkg_color" android:text="@string/hello" android:textColor="@color/text_color" />

Аналогічним чином здійснюється доступ до системних ресурсів, як ім'я пакета при цьому вказується @android:

< EditText android:id="@+id/myEditText" android:layout_width="fill_parent" android:layout_height="wrap_content" android:text="@android:string/ok"

android:textColor="@android:color/primary_text_light" />

Зверніть увагу на атрибут android: id = "@ + id / myEditText". Такий запис дозволяє привласнити ідентифікатор вашому компоненту ресурсу (в даному випадку елементу розмітки) і надалі використовувати цей ідентифікатор для отримання примірника ресурсу.

Посилання на поточні візуальні стилі дозволяють використовувати діючі зараз атрибути стилів, замість того, щоб заново їх визначати. Для вказівки посилання на такий ресурс використовується символ ? Замість @:

< EditText android:id="@+id/myEditText" android:layout_width="fill_parent" android:layout_height="wrap_content" android:hint="@string/edit_text_hint"

android:backgroundColor="?android:backgroundColor" />

Однією з основних переваг використання зовнішніх по відношенню до коду ресурсів - можливість використання механізму автоматичного вибору ресурсів. Користуючись описаним нижче механізмом, можна створювати індивідуальні ресурси для різних апаратних конфігурацій, мов, регіонів і т.д. Android під час виконання додатка сам вибере найбільш підходящі ресурси.

Для індивідуального налаштування програми доступні наступні можливості:

– MCC (Mobile Country Code) і MNC (Mobile Network Code),
– Мова і регіон. Наприклад, en-rUS для англійської мови в американському регіоні (маленька r - від «region»), ua для українського,
– Розмір екрана (small, medium або large),
– «Широкоформатність» екрана (long або notlong),
– Орієнтація екрана (port, land або square),
– Щільність пікселів на екрані (ldpi, mdpi, hdpi або nodpi),
– Тип сенсорного екрану (notouch, stylus або finger),
– Доступність клавіатури (keysexposed, keyshidden або keyssoft),
– Тип вводу (nokeys, qwerty або 12key),
– Спосіб навігації (nonav, dpad, trackball або wheel).

Альтернативні ресурси розташовуються в підкаталогах каталогу res, при цьому використовуються модифікатори стандартних імен підкаталогів з ресурсами. Наприклад, файл, що містить рядкові константи для англійської мови, буде розташовуватися за наступним шляхом: res/values-en/strings.xml модифікаторів в даному випадку є суфікс -en, доданий до імені каталога values.

Завдання для виконання

Завдання 1 «Відстеження станів Активності»

1. В наявному проекті HelloAndroidWorld відкрийте в редакторі клас HelloAndroidWorld.java.

2. Перевизначте методи onPause, onStart, onRestart для класу і внесіть зміни в метод onCreate:

```
package com.example.helloandroidworld; import android.app.Activity;
import android.os.Bundle; import android.widget.Toast;
public class HelloAndroidWorld extends Activity {
/** Called when the activity is first created. */ @Override
public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState); setContentView(R.layout.main);
Toast.makeText(this, "onCreate()", Toast.LENGTH_LONG).show();
```

```
}
@Override
protected void onPause() {
Toast.makeText(this, "onPause()", Toast.LENGTH_LONG).show();
super.onPause();
}
@Override
protected void onRestart() { super.onRestart();
Toast.makeText(this, "onRestart()", Toast.LENGTH_LONG).show();
}
@Override
protected void onStart() {
super.onStart();
Toast.makeText(this, "onStart()", Toast.LENGTH_LONG).show();}}
```

3. Запустіть додаток на виконання в емуляторі (Ctrl + F11, Android Project) і переконайтеся, що при зміні стану додатки HelloAndroidWorld на екрані емулятора з'являються відповідні повідомлення типу Toast, як показано нижче на екранній формі:

4. Поексперементуйте в додатку, щоб з'ясувати, за яких умов викликаються реалізовані обробники подій.

Завдання 2. «Використання значень рядків і кольорів»

1. В наявному проекті HelloAndroidWorld створіть файл colors в каталозі res / values: File → New → Android XML File:

2. Відредагуйте вміст файлу res / values / colors.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<color name="view_bkg_color">#FF0</color>
<color name="screen_bkg_color">#F88</color>
<color name="text_color">#8004</color>
```

</resources>

3. Відредагуйте вміст файла res/values/strings.xml:

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

<resources>

<string name="hello">Hello, Android World!</string>

<string name="app_name">HelloAndroidWorld</string>

</resources>

4. Внесіть зміни у файл розмітки res / layout / main.xml:

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

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout_width="fill_parent" android:layout_height="fill_parent" android:orientation="vertical" android:background="@color/screen_bkg_color">

<TextView

android:layout_width="fill_parent" android:layout_height="wrap_content" android:text="@string/hello" android:background="@color/view_bkg_color" android:textColor="@color/text_color"/>

</LinearLayout>

5. Збережіть всі незбережені файли і запустіть додаток.

6. Впевніться, що зовнішній вигляд програми змінився відповідно до визначених ресурсів.

7. Поекспереметуйте зі значеннями кольорів, прозорістю і HTML-тегами в строкових константах для зміни зовнішнього вигляду програми.

Завдання 3 «Локалізація додатку»

1. Змініть ресурси програми HelloAndroidWorld так, щоб на екрані відображалось HelloAndroidWorld і HelloWorld

2. Додайте потрібні підкаталоги і ресурси в каталог res проекту HelloAndroidWorld, щоб при налаштуванні української мови додаток змінював свій вигляд на наступний:

3. Зробіть ті ж дії для польської і української мови:

4. Відновіть мовні налаштування на віртуальному пристрої.

Завдання для виконання

Виконайте завдань до виконання у відповідності до теоретичних відомостей поданих вище.

Вимоги до звіту

В звіт по роботі включіть текст програм, що реалізують описані завдання та екранні форми, що відображають їх виконання.