ПРАКТИЧНА РОБОТА №6

Тема роботи: Intent в ОС Android.

Мета роботи: Ознайомитись з викликом активності за допомогою явного наміру (інтенту), з використанням неявних намірів, з одержанням даних з наміру, з використанням SharedPreferences для збереження налаштувань та зі створенням і використанням меню.

Теоретичні відомості

Наміри (Intent) в Android використовуються як механізм передачі повідомлень, який може працювати як всередині однієї програми, так і між додатками. Наміри можуть застосовуватися для:

• оголошення про бажання (необхідності) застосування запуску активності або сервісу для виконання певних дій;

• повідомлення про те, що сталась якась подія;

• явного запуску зазначеного сервісу або активності.

Останній варіант найбільш часто використовується.

Використання намірів для запуску активностей та обробка їх результатів

Щоб запустити потрібну активність, викликається метод start Activity (some Intent).

У конструкторі намірів можна явно вказати клас активності, яку потрібно запустити, або дію, яку потрібно виконати. У другому випадку система автоматично підбере потрібну активність, використовуючи механізм Intent Resolution. Метод startActivity знаходить та запускає активність, найбільш відповідну наміру користувача.

По закінченні роботи запущеної таким чином активності запустивши її активність не отримує ніяких повідомлень про результати обробки намірів. Якщо потрібно отримувати результати, використовується метод startActivityForResult.

Для явної вказівки того, яку активність (конкретний клас у додатку) потрібно запустити, створюються наміри за допомогою вказівки параметрів наступного конструктора: поточний Контекст додатка і клас Активності для запуску.

Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);

startActivity(intent);

Активність, запущена за допомогою методу startActivity, повністю незалежна від активності, що її запустила і, відповідно, завершує свою роботу, нікому про це не повідомляючи. У той же час, програміст може запускати активності, «пов'язані» зі своїм «батьком». Такий спосіб відмінно підходить для ситуацій, коли «дочірня» активність повинна обробити введення даних від користувача і надати результати обробки «батьківської» активності. Активності, що запускаються таким чином (за допомогою методу

startActivityForResult) повинні бути «зареєстровані» в файлі маніфесту додатка.

На відміну від методу startActivity, метод startActivityForResult вимагає явної вказівки ще одного параметра - коду запиту (request code). Цей параметр використовується викликаючою активністю для визначення того, яка саме дочірня активність завершила роботу і (можливо) надала результати:

```
private static final int BUY_BEER = 1;
Intent intent = new Intent(this, MyOtherActivity.class);
startActivityForResult(intent, BUY_BEER);
```

Коли дочірня активність готова до завершення роботи, до виклику методу finish потрібно викликати метод setResult для передачі результатів викликаючої активності.

Метод setResult отримує два параметри: код повернення і сам результат, представлений у вигляді намірів.

Код повернення (result code), що повертається з дочірної активності - це, зазвичай, або Activity.RESULT_OK, або Activity.RESULT_CANCELED. У випадку, якщо дочірня активність завершить роботу без виклику методу setResult, код повернення, переданий батьківській активності, дорівнюватиме Activity.RESULT_CANCELED.

У деяких випадках може знадобитися використовувати власні коди повернення для обробки певних ситуацій. Метод setResult в якості коду повернення сприймає будь – яке цілочисельне значення.

Намір, що повертається як результат, часто містить URI, який вказує на конкретний об'єкт даних, та / або набір додаткових значень, записаних у властивість наміру extras, що використовуються для передачі додаткової інформації.

Приклад вказівки обробника натискання на кнопку button, що встановлює результат роботи і завершує поточну активність:

```
Button button = (Button) findViewById(R.id.ok_button);
button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
                Intent result = new Intent();
                result.putExtra("teacher name", "Mike Varakin");
                setResult(RESULT_OK, result);
                finish();}     });
```

Коли дочірня активність завершує роботу, в батьківській активності викликається обробник onActivityResult, який отримує наступні параметри:

• Request code. Використаний при запуску дочірньої активності код запиту.

• Result code. Код повернення.

• Data. Намір, що використовується для упаковки повертаються даних.

Приклад обробника onActivityResult:

```
private static final int SELECT_VICTIM = 1;
```

```
private static final int DRINK_BEER = 2;
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
super.onActivityResult(requestCode, resultCode, data);
switch (requestCode) {
        case (SELECT_VICTIM): {
                if (resultCode == Activity.RESULT_OK) {
                        String selectedVictim = data.getStringExtra("victim");
                }
                break;
        }
case (DRINK_BEER): {
        if (resultCode == Activity.RESULT_OK) {
                // Обробити результати заказаної дії
        }
        break;      }}
```

Неявні наміри

Неявні наміри використовуються для запуску активності, для виконання зазначених дій в умовах, коли невідомо, яка саме активність (і з якого додатку) буде використовуватися.

При створенні наміру, який в подальшому буде переданий методом startActivity, необхідно призначити дію (action), яку потрібно виконати, і, можливо, вказати URI даних, які потрібно обробити. Також можна передати додаткову інформацію за допомогою властивості extras наміру. Android сам знайде підходящу активність (грунтуючись на характеристиках наміру) і запустить її. Приклад неявного виклику телефонного додзвонювача:

```
Intent intent = new Intent(Intent.ACTION_DIAL,
                Uri.parse("tel:(495)502-99-11"));
startActivity(intent);
```

Для визначення того, який саме компонент повинен бути запущений для виконання дій, зазначених у намірах, Android використовує фільтри намірів (Intent Filters). Використовуючи фільтри намірів, додатки повідомляють системі, що вони можуть виконувати певні дії (action) з певними даними (data) за певних умов (category) на замовлення інших компонентів системи.

Для реєстрації компонента додатку (активності або сервісу) в якості потенційного обробника намірів, потрібно додати елемент <intent-filter> як дочірнього елемента для потрібного компонента в маніфест файлі додатку. У елемента <intent-filter> можуть бути вказані такі дочірні елементи (і відповідні атрибути у них):

• <action>. Атрибут Android: name даного елемента використовується для зазначення назви дії, що може обслуговуватися. Кожен фільтр намірів

повинен містити не менше одного вкладеного елемента <action>. Якщо не вказати дію, жоден намір не буде «проходити» через цей фільтр. У додатку головної активності в маніфест файлі повинні бути вказаний фільтр намірів з дією android.intent.action.MAIN.

• <category>. Повідомляє системі, за яких обставин необхідно обслуговуватися дію (за допомогою атрибуту android: name). Всередині <intent-filter> може бути зазначено кілька категорій. Категорія android.intent.category.LAUNCHER вимагає активності, яка бажає мати «іконку» для запуску. Активності, що запускаються за допомогою методу startActivity, зобов'язані мати категорію android.intent.category.DEFAULT

• <data>. Дає можливість вказати тип даних, які може обробляти компонент. <intent-filter> може містити кілька елементів <data>. У цьому елементі можуть використовуватися наступні атрибути:

• android: host: ім'я хоста (наприклад, www.specialist.ru)
• android: mimetype: оброблюваний тип даних (наприклад, text / html)
• android: path: «шлях» всередині URI (наприклад, / course / android)
• android: port: порт сервера (наприклад, 80)
• android: scheme: схема URI (наприклад, http)

Приклад вказівки фільтра намірів:

```
<activity android:name=".MyActivity"
        android:label="@string/app_name" >
        <intent-filter>
          <action android:name="android.intent.action.SEND" />
          <category android:name="android.intent.category.DEFAULT" />
          <data android:mimeType="text/plain" />
        </intent-filter>
</activity>
```

При запуску активності за допомогою методу startActivity неявний намір зазвичай підходить тільки одній активності. Якщо для даного наміру підходять кілька активностей, користувачеві пропонується список варіантів.

Визначення того, які активності підходять для наміру, називається Intent Resolution. Його завдання - визначити найбільш підходящі фільтри намірів, що належать компонентам встановлених додатків. Для цього використовуються наступні перевірки в зазначеному порядку:

• Перевірка дій. Після цього кроку залишаються тільки компоненти додатків, у яких в фільтрі намірів вказано дію наміру. У випадку, якщо дія в намірі відсутня, збіг відбувається для всіх Фільтрів Намірів, у яких вказано хоча б одну дію.

• Перевірка категорій. Всі категорії, наявні у намірах, повинні бути присутніми в фільтрі намірів. Якщо у намірів немає категорій, то на даному етапі йому відповідають всі фільтри намірів, за одним винятком, згадуваним вище: активності, що запускаються за допомогою методу startActivity, зобов'язані мати категорію android.intent.category.DEFAULT, оскільки наміру, використаному в цьому випадку, за замовчуванням присвоюється дана

категорія, навіть якщо розробник не вказав нічого явно. Якщо у активності присутня дія android.intent.action.MAIN і категорія android.intent.category.LAUNCHER, йому не потрібно мати категорію android.intent.category.DEFAULT.

• Перевірка даних. Тут застосовуються такі правила:

- Намір, що не містить ні URI, ні типу даних, проходить через Фільтр, якщо він теж нічого перерахованого не містить.
- Намір, який має URI, але не містить тип даних (і тип даних неможливо визначити по URI), проходить через Фільтр, якщо URI Наміри збігається з URI Фільтра. Це справедливо лише у випадку таких URI, як mailto: або tel:, які не посилаються на реальні дані.
- Намір, що містить тип даних, але не містить URI підходить тільки для аналогічних фільтрів намірів.
- Намір, що містить і тип даних, і URI (або якщо тип даних може бути обчислений з URI), проходить цей етап перевірки, тільки якщо його тип даних присутній у фільтрі. У цьому випадку URI повинен співпадати з наміром вказаним URI виду content: або file:, а у фільтрі URI не зазначений. Тобто, передбачається, що якщо у компонента в фільтрі вказаний тільки тип даних, то він підтримує URI виду content: або file:.

У випадку, якщо після всіх перевірок залишається кілька додатків, користувачеві пропонується вибрати додаток самому. Якщо підходящих додатків не знайдено, в намірі активності виникає виняток.

Збереження стану і налаштувань додатку

Оскільки життєвий цикл додатків і активностей в Android може перерватися в будь-який момент, для підвищення привабливості та зручності користувацького інтерфейсу бажано мати можливість зберігати (і відновлювати) стан активності не тільки при виході з активного стану, а й між запусками. Android пропонує для цього наступні механізми:

• Загальні налаштування (Shared preferences): простий спосіб збереження стану UI і налаштування додатку, що використовує пари ключ / значення для зберігання примітивних типів даних і забезпечення доступу до них по імені.

• Збереження стану програми: для активностей існують обробники подій, що дозволяють зберігати і відновлювати поточний стан UI, коли виходить і повертається в активний режим. Для збереження стану використовується об'єкт класу Bundle, який передається методам onSaveInstanceState (для збереження стану), onCreate і onRestoreInstanceState (для відновлення). Для доступу до даних в цьому випадку також використовуються пари ключ / значення. Обробники подій з суперкласів беруть на себе основну роботу по збереженню та відновленню виду UI, фокусу полів і т. д.

• Пряма робота з файлами. Якщо не підходять описані вище варіанти, додаток може напряму читати і писати дані з файлу. Для цього можна використовувати стандартні класи та методи Java, що забезпечують введення / виведення, так і методи openFileInput і openFileOutput, надані Android, для спрощення читання і запису потоків, що відносяться до локальних файлів.

Клас SharedPreferences пропонує методи для збереження й отримання даних, що записуються у файли, доступні по замовчуванню тільки конкретному додатку. Такий спосіб зберігання забезпечує збереження цих даних не тільки в перебігу життєвого циклу додатка, але і між запусками і навіть перезавантаженням ОС.

Для збереження даних у файлі використовується транзакційний механізм: спочатку потрібно отримати об'єкт класу SharedPreferences.Editor для конкретного файлу налаштувань, після чого за допомогою методів виду put тип цього об'єкта та встановити потрібні значення. Запис значень проводиться методом commit.

Наведемо приклад збереження даних:

```
private static final String PREFS = "PREFS";
static final String KEY_STATION = "selectedStation";
private SharedPreferences prefs;
private static final String NOTHING_SELECTED = "Ничего не выбрано";
private String selectedStation;
prefs = getSharedPreferences(PREFS, MODE_PRIVATE);
Editor editor = prefs.edit();
editor.putString(KEY_STATION, selectedStation);
editor.commit();
```

Наведемо приклад отримання даних:

```
prefs = getSharedPreferences(PREFS, MODE_PRIVATE);
selectedStation = prefs.getString(KEY_STATION, NOTHING_SELECTED);
tv.setText(selectedStation);
```

Робота з файлами

Методи openFileInput і openFileOutput дають можливість працювати тільки з файлами, що знаходяться в «персональному» каталозі додатку. Як наслідок, вказівка роздільників ("/") в імені файлу призведе до викиду винятку.

За замовчуванням файли, відкриті на запис, перезаписуються. Якщо це не те, що потрібно, при відкритті встановлюється режим MODE_APPEND.

Приклад роботи з файлами:

```
String FILE_NAME = "app_data";
// Відкриття вихідного файлового потоку
FileOutputStream fos = openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
// Відкриття вхідного файлового потоку
FileInputStream fis = openFileInput(FILE_NAME);
```

String FILE_NAME = "app_data";

Через об'єкт Context в додатку також можливий доступ до двох корисних методів:

• fileList - повертає список файлів додатку;

• deleteFile видаляє файл з каталогу додатку.

Якщо додатку необхідно мати доступ до інформації, яку незручно зберігати, наприклад, в СУБД, ці дані можна використовувати у вигляді «сирих» ресурсів, записавши їх у файли в каталозі res / raw. Яскравий приклад подібних даних - словники.

Статичні файли, як випливає з назви, доступні тільки для читання, для їх відкриття використовується метод openRawResource:

Resources res = getResources();

InputStream file = res.openRawResource(R.raw.filename);

Робота із меню в Android

Використання меню в додатках дозволяє зберегти цінний екранний простір, який в іншому випадку було б зайнято відносно рідко використовуваними елементами інтерфейсу.

Кожна активність може мати меню, що реалізують специфічні функції. Можна використовувати також контекстні меню, індивідуальні для кожного подання на екрані.

В Android реалізована підтримка триступеневої системи меню, оптимізована, в першу чергу, для невеликих екранів:

• Основне меню розміщується внизу на екрані при натисканні на кнопку «menu» пристрою. Воно може відображати текст і іконки для обмеженого (за замовчуванням, не більше шести) числа пунктів. Для цього меню рекомендується використовувати іконки з колірною гамою у вигляді відтінків сірого з елементами рельєфності. Це меню не може містити радіокнопки і чекбокси. Якщо число пунктів такого меню перевищує максимально допустиме значення, в меню автоматично з'являється пункт з написом «ще» («more»). При натисканні на нього відобразиться Розширене меню.

• Розширене меню відображає прокручуваний список, елементами якого є пункти, що не увійшли до основного меню. У цьому списку не можуть відображатися іконки, але є можливість відображення радіокнопок і чекбоксів. Оскільки не існує способу відобразити розширене меню замість основного, про зміну стану якихось компонентів програми або системи рекомендується повідомляти користувача за допомогою зміни іконок або тексту пунктів меню.

• Дочірнє меню (меню третього рівня) може бути викликано з основного або розширеного меню і відображається у спливаючому вікні. Вкладеність не підтримується, і спроба викликати з дочірнього ще одне меню призведе до викиду винятку.

При зверненні до меню викликається метод onCreateOptionsMenu активності та для появи меню на екрані його потрібно перевизначити. Даний метод отримує параметр об'єкт класу Menu, який надалі використовується для маніпуляцій з пунктами меню.

Для додавання нових пунктів в меню використовується метод add об'єкта Menu з наступними параметрами:

• Група об'єднання пунктів меню для групової обробки.

• Ідентифікатор - унікальний ідентифікатор пункту меню. Цей ідентифікатор передається оброблювачу натискання на пункт меню - методу onOptionsItemSelected.

• Порядок - значення, яке вказує порядок у якому пункти меню будуть виводитися.

• Текст - напис на пункті меню.

Після успішного створення меню метод onCreateOptionsMenu повинен повернути значення true.

Наведений нижче приклад показує створення меню з трьох пунктів з використанням строкових ресурсів:

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    menu.add(0, Menu.FIRST, Menu.NONE, R.string.menu_item1);
    menu.add(0, Menu.FIRST+1, Menu.NONE, R.string.menu_item2);
    menu.add(0, Menu.FIRST+2, Menu.NONE, R.string.menu_item3);
    return true; }
```

Для пошуку пунктів меню за ідентифікатором можна використовувати метод findItem об'єкта Menu.

Найбільш корисними параметрами пунктів меню є наступні:

• Короткі заголовки, які використовуються у випадку, якщо пункт може відобразитися в основному меню. Встановлюється методом setTitleCondensed об'єкта класу MenuItem:

```
menuItem.setTitleCondensed ("заголовок");
```

• Іконки - ідентифікатор Drawable, що містить потрібну картинку:

```
menuItem.setIcon (R.drawable.menu_item_icon);
```

• Оброблювач вибору пункту меню який можна встановити, але не рекомендується з міркувань підвищення продуктивності, краще використовувати обробник всього меню (onOptionsItemSelected). Тим не менш, приклад:

```
menuItem.setOnMenuItemClickListener(new OnMenuItemClickListener() {
    public boolean onMenuItemClick(MenuItem _menuItem) {
    // обробити вибір пункту
    return true;
    }
});
```

• Намір - автоматично передається методу startActivity, якщо відбувається натиснення на пункт меню, який не було оброблено onMenuItemClickListener і onOptionsItemSelected:

menuItem.setIntent(new Intent(this, MyOtherActivity.class));

Безпосередньо перед виведенням меню на екран викликається метод onPrepareOptionsMenu поточної Активності, і перевизначаючи його можна динамічно змінювати стан пунктів меню: дозволяти / забороняти, робити невидимим, змінювати текст і т. д.

Для пошуку пункту меню, що підлягає модифікації можна використовувати метод findItem об'єкта Menu, переданого як параметр:

```
@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    MenuItem menuItem = menu.findItem(MENU_ITEM);
    //модифікувати пункт меню....
    return true;
}
```

Android дозволяє обробляти всі пункти меню (вибирати їх) одним обробником onOptionsItemSelected. Вибраний пункт меню передається цьому оброблювачу в якості об'єкта класу MenuItem.

Для реалізації потрібної реакції на вибір пункту меню потрібно визначити, що саме було обрано. Для цього використовується метод getItemId переданий як параметр об'єкта, а отриманий результат порівнюється з ідентифікаторами, використаними при додаванні пунктів у меню в методі onCreateOptionsMenu.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        // Перевірити кожен відомий пункт
        case (MENU_ITEM):
            // зробити що-небудь...
            return true;
    }
    return super.onOptionsItemSelected(item);
}
```

При появі на екрані, дочірні та контекстні меню виглядають однаково, у вигляді плаваючих вікон, але при цьому створюються по-різному.

Для створення дочірніх меню використовується метод addSubMenu об'єкта класу Menu:

```
SubMenu sub = menu.addSubMenu(0, 0, Menu.NONE, "дочірнє меню");
sub.setHeaderIcon(R.drawable.icon);
sub.setIcon(R.drawable.icon);
MenuItem submenuItem = sub.add(0, 0, Menu.NONE, "пункт дочірнього меню");
```

Як вже було сказано вище, вкладені дочірні меню Android не підтримує.

Найбільш поширеним способом створення контекстного меню в Android є перевизначення методу onCreateContextMenu активності:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenu.ContextMenuInfo menuInfo) {
super.onCreateContextMenu(menu, v, menuInfo);
menu.setHeaderTitle("Конекстне меню");
menu.add(0, Menu.FIRST, Menu.NONE, "Пункт 1");
menu.add(0, Menu.FIRST+1, Menu.NONE, "Пункт 2");
menu.add(0, Menu.FIRST+2, Menu.NONE, "Пункт 3");}
```

Реєстрація обробника контекстного меню для потрібних View здійснюється за допомогою методу registerForContextMenu:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        tv = (TextView) findViewById(R.id.text_view);
        registerForContextMenu(tv);
}
```

Приклад обробника:

```
@Override
public boolean onContextItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case DELETE_ID:
                AdapterContextMenuInfo info = (AdapterContextMenuInfo)
        item
                                .getMenuInfo();
                db.deleteItem(info.id);
                populate();
                return true;
        }
        return super.onContextItemSelected(item);
}
```

Часто буває найзручніше описувати меню, в тому числі ієрархічні, у вигляді ресурсів. Про переваги такого підходу говорилося вище.

Меню традиційно описуються і зберігаються в каталозі res / menu проекту. Всі ієрархії меню (якщо є ієрархічні меню) повинні знаходитися в окремих файлах, а ім'я файлу буде використовувати як ідентифікатор ресурсу. Кореневим елементом файлу повинен бути тег <menu>, а пункти меню описуються тегом <item>. Властивості пунктів меню описуються відповідними атрибутами:

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

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/item01"
        android:icon="@drawable/menu_item"
        android:title="Пункт 1">
    </item>
    <item
        android:id="@+id/item02"
        android:checkable="true"
        android:title="Пункт 2">
    </item>
    <item
        android:id="@+id/item03"
        android:title="Пункт 3">
    </item>
    <item
        android:id="@+id/item04"
        android:title="Дочірнє меню 1">
        <menu>
        <item
            android:id="@+id/sub1item01"
            android:title="Пункт дочірнього меню 1">
        </item>
    </menu>
</item>
<item
    android:id="@+id/item05"
    android:title="Дочірнє меню 2">
    <menu>
        <item
            android:id="@+id/sub2item01"
            android:title="Пункт дочірнього меню 2">
        </item>
    </menu>
  </item>
</menu>
```

Для створення об'єктів Menu з ресурсів в події onCreateOptionsMenu і onCreateContextMenu використовується метод inflate об'єкта типу MenuInflater:

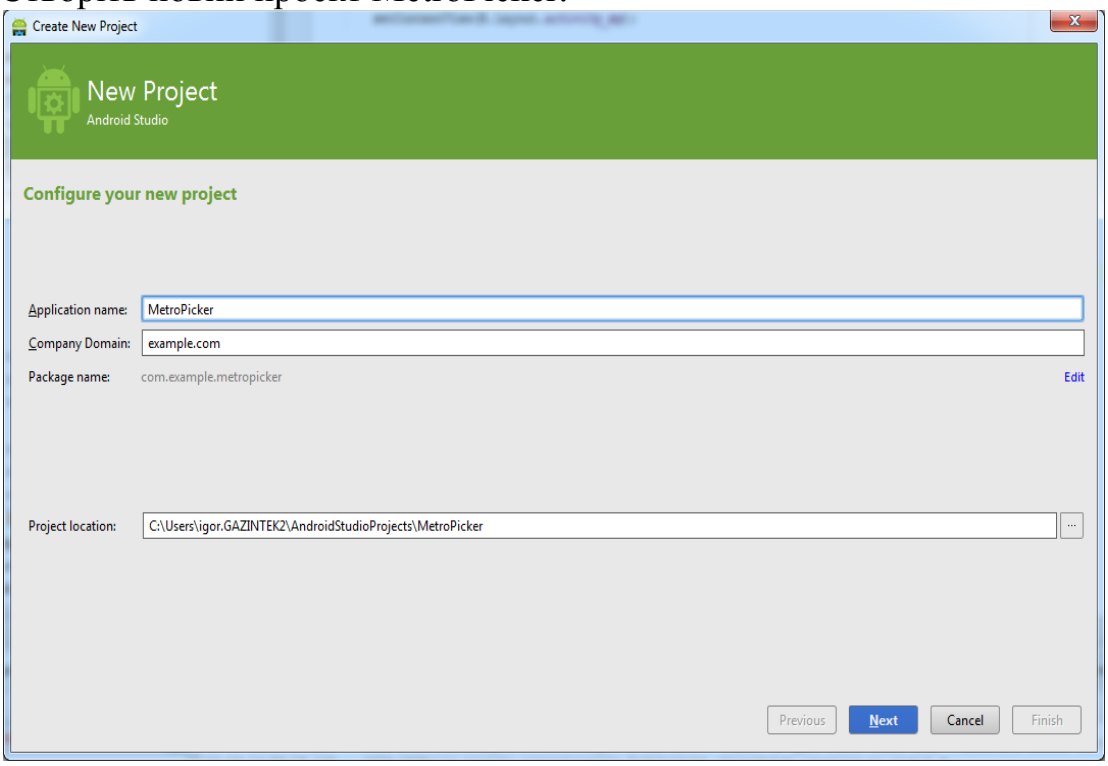
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    super.onCreateOptionsMenu(menu);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu1, menu);
```

```
        return true;
    }
```

Порядок виконання роботи

Завдання 1 «Виклик активності за допомогою явного наміру та отримання результатів роботи»

1. Створіть новий проект MetroPicker.



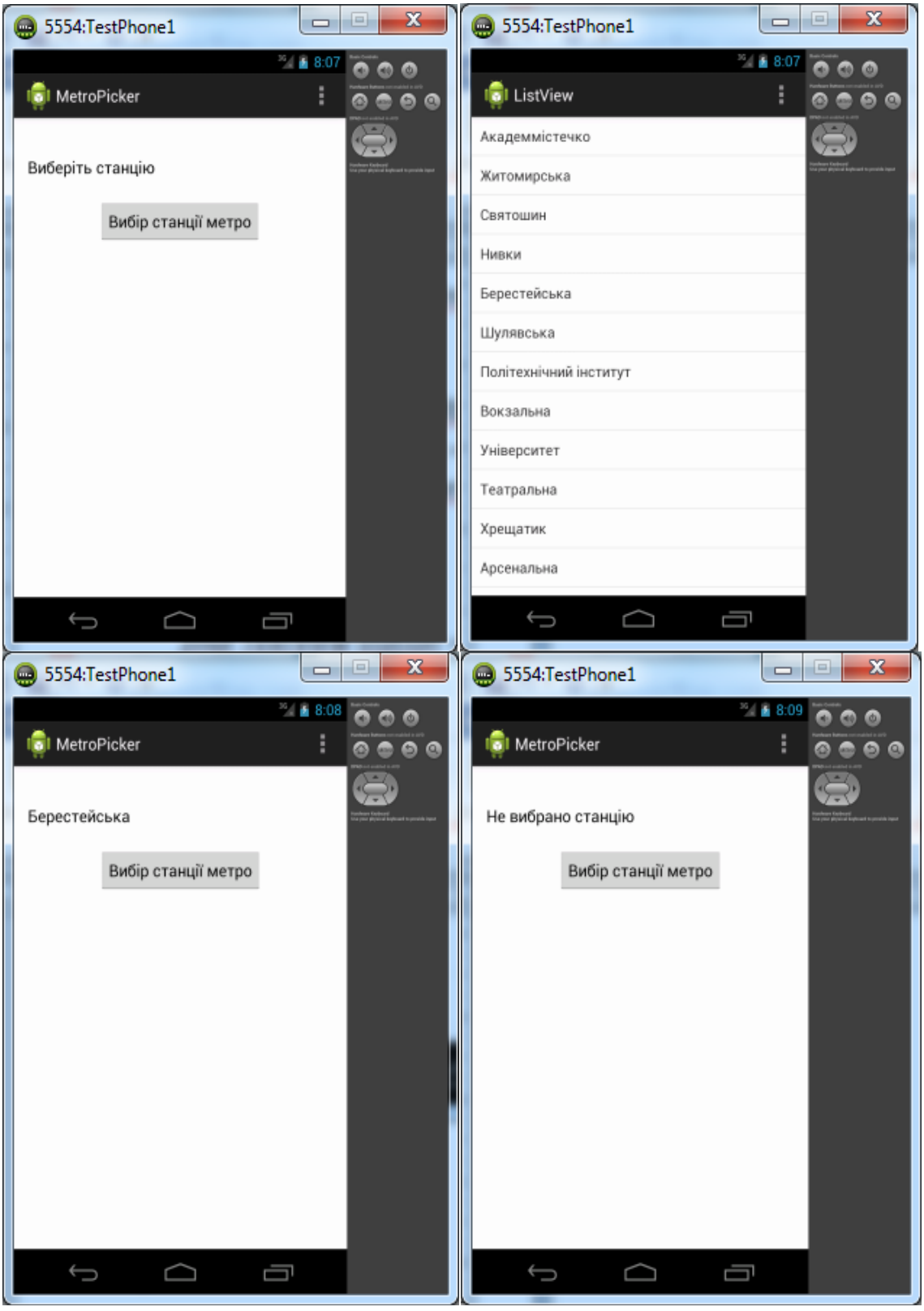
2. Додайте допоміжну активність ListView Activity для відображення і вибору станцій метро.

3. Відредагуйте файл розмітки res / layout / main.xml: додайте кнопку вибору станції метро, присвоївши ідентифікатори віджетам TextView і Button для того, щоб на них можна було посилатися в коді.

4. Встановіть обробник натискання на кнопку в головній активності для виклику списку станції і вибору потрібної станції.

5. Напішіть потрібний обробник для установки обраної станції метро в віджет TextView батьківської активності (метод setText віджета TextView дозволяє встановити відображуваний текст). Не забудьте обробити ситуацію, коли користувач натискає кнопку «Назад» (в цьому випадку «ніякої станції не вибрано» і головна активність повинна сповістити про це користувача).

6. Впевніться в працездатності створеного додатка, перевіривши реакцію на різні дії потенційних користувачів.



Завдання 2 «Використання неявних намірів»

1. Змініть проект MetroPicker так, щоб для запуску активності ListView Activity використовувався неявний намір з дією (action), визначеною у вашому додатку і має значення "com.example.metropicker. intent.action.PICK_METRO_STATION".

2. Перевірте роботу програми. Визначення наміру, що викликав запуск активності. Оскільки об'єкти типу Intent служать, в тому числі, для передачі інформації між компонентами одного або декількох додатків, може виникнути необхідність у роботі з об'єктом наміру, що викликав активність.

Для отримання доступу до цього об'єкта використовується метод getIntent. приклад:

Intent intent = getIntent();

Маючи даний об'єкт, можна отримати доступ до інформації, що міститься в ньому:

• метод getAction повертає дію наміру;

• метод getData повертає дані наміру (зазвичай URI);

• набір методів для різних типів виду getТИПExtra дозволяє отримати доступ до типізованих значеннь, що зберігаються у властивості extras наміру.

Приклади:

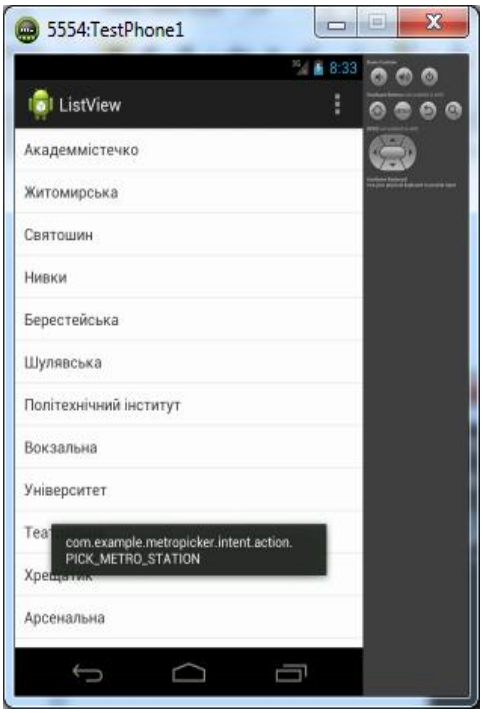
String action = intent.getAction();

Uri data = intent.getData();

Задання 3 «Одержання даних з наміру»

1. Модифікуйте методи onCreate з попередньої лабораторної роботи так, щоб за допомогою Toast вони показували дію викликаних їх намірів.

2. Перевірте роботу програми:



Завдання 4 «Використання SharedPreferences для збереження стану»

1. Модифікуйте методи onCreate і onActivityResult проекту MetroPicker для збереження обраної станції метро між запусками програми.

2. Перевірте працездатність програми.

Завдання 5 «Використання SharedPreferences для збереження налаштувань»

1. Модифікуйте проект Controls Sample так, щоб стан керуючих елементів зберігався і відновлювався між запусками програми.

2. Перевірте працездатність програми.

Завдання 6 «Створення і використання меню»

Модифікуйте проект MetroPicker наступним чином:

1. Додайте головне меню в Активність, відображаючи список станцій метро. У меню має бути один пункт «повернутися». Меню створіть динамічно в коді, без використання строкових ресурсів.

2. Динамічно створіть контекстне меню для TextView, що буде відображати обрану станцію метро головної активності. Вибір пункту меню повинен скидати обрану станцію.

3. Для головної активності створіть основне меню з двох пунктів: «скинути» і «вийти». Реалізуйте потрібні функції при виборі цих пунктів. Повторіть реалізацію п.п. 1, 2 і 3 за допомогою ресурсів, що описують меню.

Завдання для виконання

Виконайте завданння у відповідності до теоретичних відомостей поданих вище.

Вимоги до звіту

В звіт по роботі включіть текст програм, що реалізують описані завдання та екранні форми (скрін-шоти), що відображають їх виконання.

Контрольні запитання

1. Для чого використовують меню у додатку?
2. Який метод використовується для створння дочірніх меню?
3. Чи вірно це? Дані типу Calendar можна безпосередньо зберегти у екземплярі класу SharedPreferences.
4. Які типи даних підтримуються класом SharedPreferences?
5. Яка різниця між явним і неявним наміром?