Лабораторна робота №11. Фрагменти

Мета: вивчити порядок роботи з компонентом Fragment в Android.

Теоретичні відомості

Фрагмент (Fragment) – це частина користувацького інтерфейса в activity. Активність може складатися з декількох фрагментів для побудови динамічного інтерфейса (рис. 1).

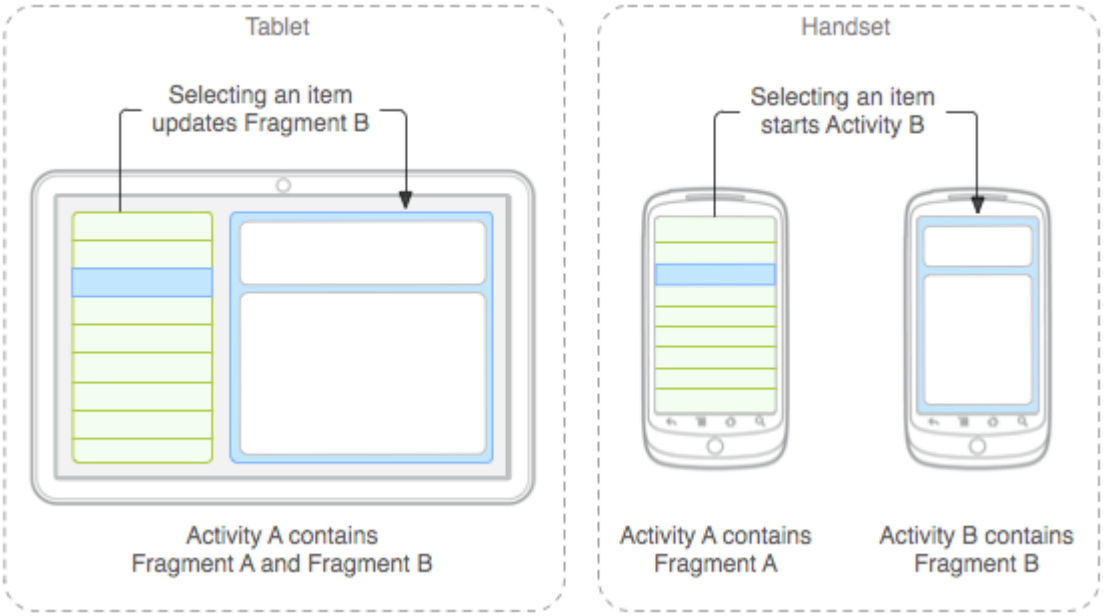

Рис. 1

Фрагмент можна розглядати як модульну частину активності, яка має свій життєвий цикл та може самостійно обробляти події вводу даних. Також фрагменти можна додавати та знищувати безпосередньо під час виконання activity.

Фрагмент завжди повинен бути вбудованим в активність, а на його життєвий цикл впливає життєвий цикл активності. Наприклад, коли активність призупинена, в цьому ж стані знаходяться і всі фрагменти цієї активності. Але коли activity має стан resumed, можна маніпулювати з кожним фрагментом окремо.

Перевагою фрагментів є можливість їх повторного використання у різних активностях. Через це, неприпустимо, щоб один фрагмент використовувався іншим.

Для створення фрагменту необхідно визначити об’єкт класа **Fragment** та описати методу зворотного виклику, які у більшості аналогічні методам зворотного виклику activity (рис. 2).

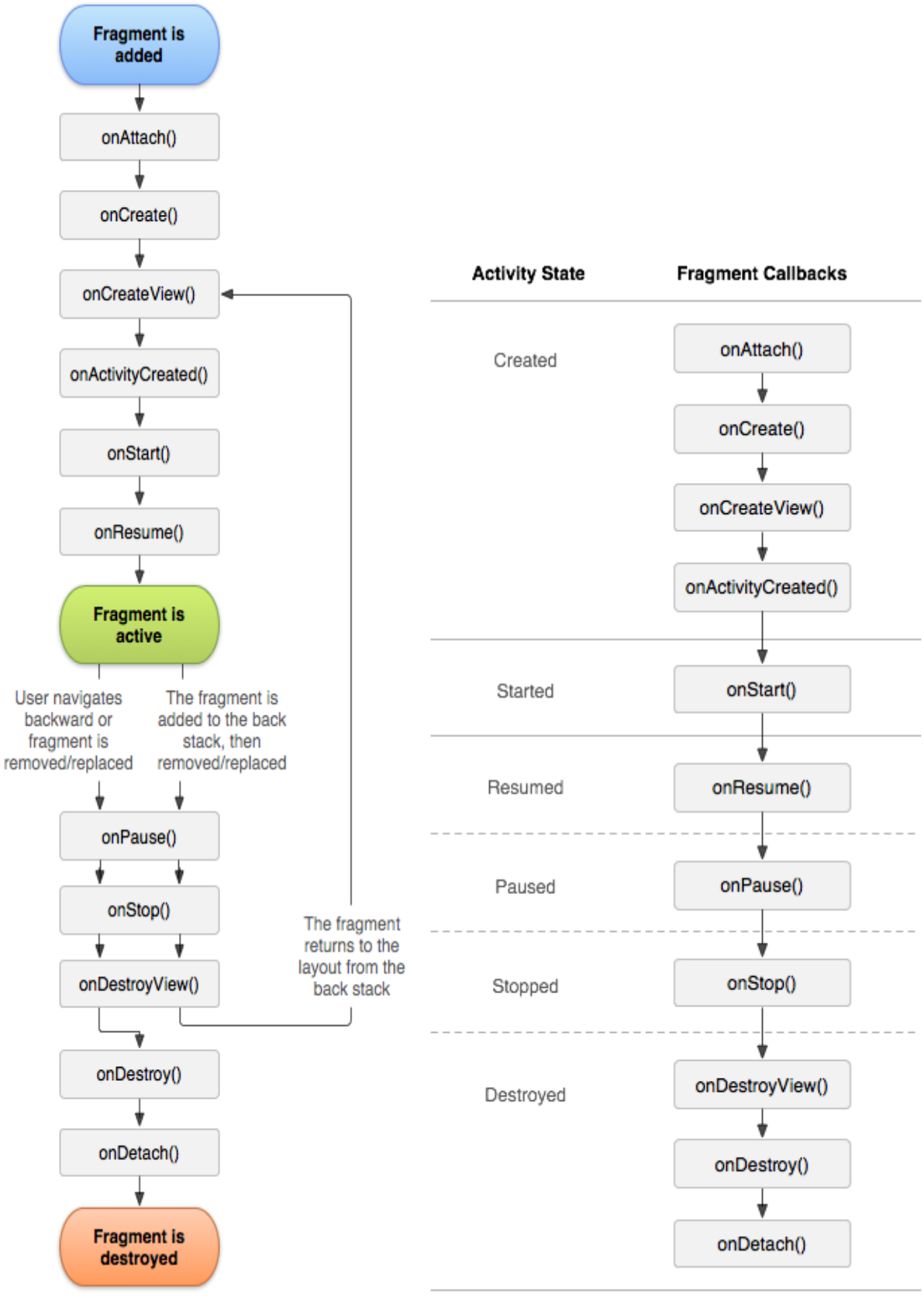

Рис. 2

Методи життєвого циклу.

Життєвий цикл фрагментів пов'язано з життєвим циклом activity, до якої відноситься фрагмент (рис. 2).

onCreate(). Система викликає цей метод при створенні фрагмента. В цьому методі ініціалізуються компоненти фрагмента.

onCreateView(). Система викликає цей метод при першому відображенні фрагмента на екрані. Для прорисовки користувацького інтерфейса потрібно вернути в цьому методі об'єкт View, який є кореневим в макеті фрагмента.

onPause(). Система викликає цей метод як першу ознаку того, що користувач збирається перейти з фрагмента. Зазвичай в цьому методі потрібно зберігати той стан, як потім повинен бути відтворений після перезапуску фрагмента.

onAttach(). Викликається коли фрагмент пов'язується з активністю (йому передається об'єкт activity).

onActivityCreated(). Викликається коли метод **onCreate()**, що відноситься до activity, повертає керування.

onDetach(). Викликається при розриві зв'язку між фрагментом та activity.

Створення користувацького інтерфейса. Фрагменти зазвичай використовуються як частина користувацького інтерфейса, при цьому в activity додається макет (layout) фрагмента. Це можна зробити наступним чином:

```
public static class ExampleFragment extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.example_fragment, container, false);
    }
}
```

Рис. 3

Метод **inflate()** у прикладі вище приймає три аргумента:

- Ідентифікатор ресурса макет, який потрібно відобразити.
- Об'єкт класа **ViewGroup**, кий повинен стати батьківським елементом для макета фрагмента.
- Логічне значення, яке показує, чи слід прикріпити макет фрагмента до об'єкта **ViewGroup**.

Додавання фрагмента в activity. Зазвичай, фрагмент додає частину користувацького інтерфейса в activity і цей інтерфейс вбудовується в загальну ієрархію компонентів активності.

Для розробника є **дві можливості** додати фрагмент в макет активності.

Визначити фрагмент в файлі макета активності. В цьому випадку можна вказати властивості макета фрагмента в xml файлі відповідної активності (рис. 4).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:name="com.example.news.ArticleListFragment"
            android:id="@+id/list"
            android:layout_weight="1"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
            android:id="@+id/viewer"
            android:layout_weight="2"
            android:layout_width="0dp"
            android:layout_height="match_parent" />
</LinearLayout>
```

Рис. 4

Додати фрагмент в існуючий об'єкт ViewGroup у програмному коді. Для виконання транзакцій з фрагментами (додавання, видалення, заміна фрагмента) необхідно використовувати API-інтерфейси з класа **FragmentTransaction**. Визначити екземпляр класа **FragmentTransaction** можна наступним чином:

```
FragmentManager fragmentManager = getFragmentManager()
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Рис. 5

Після цього, можна додати фрагмент методом **add()**, вказавши фрагмент, що додається та об'єкт **ViewGroup** у який додається фрагмент (рис. 6).

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

Рис. 6

Перший аргумент метода **add()** – це контейнерний об’єкт **ViewGroup** для фрагмента, який вказано за допомогою ідентифікатора ресурса. Другий аргумент – фрагмент, який необхідно додати. Метод **commit()** необхідний для коректного завершення транзакції.

Додавання фрагмента без користувацького інтерфейса. Фрагменти можуть використовуватись виконання дій в фоновому режимі. Для додавання фрагменту без користувацького інтерфейса використовується метод **add(Fragment, String)**. В методі **add(Fragment, String)** передається унікальний строковий параметр («тег»). Фрагмент буде додано, але, оскільки від не пов’язаний з елементом View, він не буде приймати виклик метода **onCreateView()**. Тому, в реалізації цього метода немає необхідності.

Якщо у фрагмента немає користувацького інтерфейса, строковий тег є єдиним засобом його ідентифікації.

Приклад активності, яка використовує фрагмент як фоновий поток без користувацького інтерфейса, наведено в прикладі **FragmentRetainInstance.java**.

```
/*
 * Copyright (C) 2010 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package com.example.android.apis.app;

import com.example.android.apis.R;

import android.app.Activity;
import android.app.Fragment;
import android.app.FragmentManager;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.ProgressBar;

/**
```

```
 * This example shows how you can use a Fragment to easily propagate state
 * (such as threads) across activity instances when an activity needs to be
 * restarted due to, for example, a configuration change.  This is a lot
 * easier than using the raw Activity.onRetainNonConfiguratinInstance() API.
 */
public class FragmentRetainInstance extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // First time init, create the UI.
        if (savedInstanceState == null) {
            getFragmentManager().beginTransaction().add(android.R.id.content,
                    new UiFragment()).commit();
        }
    }

    /**
     * This is a fragment showing UI that will be updated from work done
     * in the retained fragment.
     */
    public static class UiFragment extends Fragment {
        RetainedFragment mWorkFragment;

        @Override
        public View onCreateView(LayoutInflater inflater, ViewGroup container,
                                 Bundle savedInstanceState) {
            View v = inflater.inflate(R.layout.fragment_retain_instance,
container, false);

            // Watch for button clicks.
            Button button = (Button)v.findViewById(R.id.restart);
            button.setOnClickListener(new OnClickListener() {
                public void onClick(View v) {
                    mWorkFragment.restart();
                }
            });

            return v;
        }

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

            FragmentManager fm = getFragmentManager();

            // Check to see if we have retained the worker fragment.
            mWorkFragment = (RetainedFragment)fm.findFragmentByTag("work");

            // If not retained (or first time running), we need to create it.
            if (mWorkFragment == null) {
                mWorkFragment = new RetainedFragment();
                // Tell it who it is working with.
                mWorkFragment.setTargetFragment(this, 0);
                fm.beginTransaction().add(mWorkFragment, "work").commit();
            }
        }

    }

    /**
     * This is the Fragment implementation that will be retained across
     * activity instances.  It represents some ongoing work, here a thread
     * we have that sits around incrementing a progress indicator.
     */
```

```
public static class RetainedFragment extends Fragment {
    ProgressBar mProgressBar;
    int mPosition;
    boolean mReady = false;
    boolean mQuiting = false;

    /**
     * This is the thread that will do our work.  It sits in a loop running
     * the progress up until it has reached the top, then stops and waits.
     */
    final Thread mThread = new Thread() {
        @Override
        public void run() {
            // We'll figure the real value out later.
            int max = 10000;

            // This thread runs almost forever.
            while (true) {

                // Update our shared state with the UI.
                synchronized (this) {
                    // Our thread is stopped if the UI is not ready
                    // or it has completed its work.
                    while (!mReady || mPosition >= max) {
                        if (mQuiting) {
                            return;
                        }
                        try {
                            wait();
                        } catch (InterruptedException e) {
                        }
                    }

                    // Now update the progress.  Note it is important that
                    // we touch the progress bar with the lock held, so it
                    // doesn't disappear on us.
                    mPosition++;
                    max = mProgressBar.getMax();
                    mProgressBar.setProgress(mPosition);
                }

                // Normally we would be doing some work, but put a kludge
                // here to pretend like we are.
                synchronized (this) {
                    try {
                        wait(50);
                    } catch (InterruptedException e) {
                    }
                }
            }
        }
    };

    /**
     * Fragment initialization.  We way we want to be retained and
     * start our thread.
     */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Tell the framework to try to keep this fragment around
        // during a configuration change.
        setRetainInstance(true);

        // Start up the worker thread.
```

```
            mThread.start();
        }

        /**
         * This is called when the Fragment's Activity is ready to go, after
         * its content view has been installed; it is called both after
         * the initial fragment creation and after the fragment is re-attached
         * to a new activity.
         */
        @Override
        public void onActivityCreated(Bundle savedInstanceState) {
            super.onActivityCreated(savedInstanceState);

            // Retrieve the progress bar from the target's view hierarchy.
            mProgressBar =
(ProgressBar)getTargetFragment().getView().findViewById(
                    R.id.progress_horizontal);

            // We are ready for our thread to go.
            synchronized (mThread) {
                mReady = true;
                mThread.notify();
            }
        }

        /**
         * This is called when the fragment is going away.  It is NOT called
         * when the fragment is being propagated between activity instances.
         */
        @Override
        public void onDestroy() {
            // Make the thread go away.
            synchronized (mThread) {
                mReady = false;
                mQuiting = true;
                mThread.notify();
            }

            super.onDestroy();
        }

        /**
         * This is called right before the fragment is detached from its
         * current activity instance.
         */
        @Override
        public void onDetach() {
            // This fragment is being detached from its activity.  We need
            // to make sure its thread is not going to touch any activity
            // state after returning from this function.
            synchronized (mThread) {
                mProgressBar = null;
                mReady = false;
                mThread.notify();
            }

            super.onDetach();
        }

        /**
         * API for our UI to restart the progress thread.
         */
        public void restart() {
            synchronized (mThread) {
                mPosition = 0;
                mThread.notify();
```

```
            }
        }
    }
}
```

Управління фрагментами. Для управління фрагментами в activity використовується клас **FragmentManager**. Щоб отримати його, потрібно викликати метод **getFragmentManager()** з коду activity.

Нижче перераховані дії, які дозволяє виконувати **FragmentManager**.

- Отримувати фрагменти, які є в активності, за допомогою метода **findFragmentById()** (для фрагментів, які мають користувацький інтерфейс) або **findFragmentByTag()** (для всіх фрагментів).
- Видаляти фрагменти з стека переходів назад методом **popBackStack()** (імітується натиснення кнопки «Назад» на пристрої).
- Регіструвати процесс-listener змін в стеку переходів назад за допомогою метода **addOnBackStackChangedListener()**.

Транзакції з фрагментами. Перевагою використання фрагментів є можливість їх створення, видалення та інших дій у відповідь на певну поведінку користувача. Будь-який набір змін у activity називається транзакція. Транзакцію можна виконати за допомогою API інтерфейсів класа **FragmentTransaction.** Кожну транзакцію можна зберегти у стеку переходів назад, яким керує activity.

Екземпляр класа **FragmentTransaction** можна отримати від **FragmentManager** наступним чином:

```
FragmentManager fragmentManager = getFragmentManager();
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
```

Можна вказати всі зміни, які необхідно виконати над фрагментом в даній транзакції, для цього існують методи **add()**, **remove()** та **replace()**. Щоб застосувати транзакцію до activity застосовується метод **commit()**.

addToBackStack() дозволяє додати транзакцію в стек переходу назад.

Приклад нижче демонструє як можна замінити один фрагмент іншим, при збереженні попереднього стану в стек переходу назад.

```
// Create new fragment and transaction
Fragment newFragment = new ExampleFragment();
FragmentTransaction transaction = getFragmentManager().beginTransaction();

// Replace whatever is in the fragment_container view with this fragment,
// and add the transaction to the back stack
transaction.replace(R.id.fragment_container, newFragment);
transaction.addToBackStack(null);

// Commit the transaction
transaction.commit();
```

Рис. 7

Якщо в транзакцію додати деякі методи (наприклад, ще раз викликати **add()** або **remove()**), а потім викликати метод **addToBackStack()**, всі зміни, які були описані до метода **commit()** будуть додані в стек переходів назад як одна транзакція.

Виклик метода **commit()** не призводить до миттєвого виконання транзакції. Вона буде запланована в потоці графічного інтерфейса. При необхідності можна викликати метод **executePendingTransactions()** для того, щоб транзакція була виконана миттєво.

Взаємодія з activity. Зазвичай екземпляр фрагмента пов'язано з певною активністю. Так, фрагмент може звернутися до свого **activity** за допомогою метода **getActivity()** та отримати, наприклад, ідентифікатор макета:

```
View listView = getActivity().findViewById(R.id.list);
```

Аналогічним чином, активність може викликати методи фрагмента, отримавши посилання на об'єкт Fragment від FragmentManager за допомогою метода findFragmentById() ало findFragmentByTag().

```
ExampleFragment fragment = (ExampleFragment)
getFragmentManager().findFragmentById(R.id.example_fragment);
```

Розглянемо приклад. Нехай додаток містить два фрагменти в одній activity: один для відображення списку статей (фрагмент А), а інший – для відображення статті (фрагмент В). Тоді фрагмент А повинен повідомляти активність про обраний пункт списка статей, а activity повідомляє фрагмент В, яку статтю потрібно відобразити.

У фрагменті А визначається інтерфейс **OnArticleSelectedListener**:

```
public static class FragmentA extends ListFragment {
    ...
    // Container Activity must implement this interface
    public interface OnArticleSelectedListener {
        public void onArticleSelected(Uri articleUri);
    }
    ...
}
```

Рис. 8

Тоді операція, яка містить цій фрагмент, реалізує інтерфейс **OnArticleSelectedListener** і перевизначає метод **onArticleSelected()** для того, щоб повідомляти фрагмент B про подію, яка відбувається в фрагменті A.

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onAttach(Activity activity) {
        super.onAttach(activity);
        try {
            mListener = (OnArticleSelectedListener) activity;
        } catch (ClassCastException e) {
            throw new ClassCastException(activity.toString() + " must implement
OnArticleSelectedListener");
        }
    }
    ...
}
```

Якщо в активності не реалізується інтерфейс, фрагмент генерує виключання **ClassCastException**. У випадку успіху елемент mListener буде містити посилання на реалізацію інтерфейса **OnArticleSelectedListener** в активності.

Якщо фрагмент A наслідує клас ListFragment, то кожний раз, коли користувач натискає елемент списка, система викликає метод onListItemClick(). Цей метод у свою чергу викликає метод onArticleSelected().

```
public static class FragmentA extends ListFragment {
    OnArticleSelectedListener mListener;
    ...
    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {
        // Append the clicked item's row ID with the content provider Uri
        Uri noteUri = ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id);
        // Send the event and Uri to the host activity
        mListener.onArticleSelected(noteUri);
    }
    ...
}
```

Рис. 9

Параметр **id**, який передається методу **onListItemClick()** – це ідентифікатор строки з обраним елементом списка, який activity використовує для отримання статті від компонента **ContentProvider** додатка.

Завдання до лабораторної роботи

1. У додатку сортування чисел з лабораторної роботи №7 створить додаткове activity яке викликається після натискання кнопки «Help».
2. У новому activity створіть два фрагмента. Перший фрагмент містить перелік 5-ти алгоритмів сортування. Другий фрагмент відображає теоретичні відомості про обраний алгоритм. При цьому, у landscape режимі відображаються обидва фрагменти одночасно, а у portrait – почергово (рис. 1).
3. При переході між різними activity та при виклику різних методів життєвого циклу активностей\фрагментів, відображати toast повідомлення з назвою активності\фрагмента та викликаного метода життєвого циклу.

Контрольні запитання

1. Що таке фрагмент?
2. Чим фрагмент відрізняється від activity?
3. Назвіть методи життєвого циклу фрагментів.