6 Багатопотоковість, AsyncTask

6.1 Клас Thread. Інтерфейс Runnable

Коли ми запускаємо програму на Android, система створює потік, який називається основним потоком програми або UI-потік. Цей потік обробляє всі зміни та події інтерфейсу користувача. Однак для допоміжних операцій, таких як надсилання або завантаження файлу, тривалі обчислення тощо, ми можемо створювати додаткові потоки.

Для створення нових потоків нам доступний стандартний функціонал класу **Thread** з базової бібліотеки Java з пакета `java.util.concurrent`, які особливої труднощі не представляють. Тим не менш, труднощі можуть виникнути при оновленні візуального інтерфейсу з потоку.

Наприклад, створимо найпростіший додаток із використанням потоків. Визначимо наступну розмітку інтерфейсу у файлі **activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="22sp"
        app:layout_constraintBottom_toTopOf="@id/button"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Запустить поток"
        app:layout_constraintTop_toBottomOf="@id/textView"
        app:layout_constraintLeft_toLeftOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Тут визначено кнопку для запуску фонового потоку, а також текстове поле для відображення деяких даних, які будуть генеруватися в запущеному потоці.

Далі визначимо в класі **MainActivity** наступний код:

```
1   package com.example.threadapp;
2
3   import androidx.appcompat.app.AppCompatActivity;
4
5   import android.os.Bundle;
6   import android.view.View;
7   import android.widget.Button;
8   import android.widget.TextView;
9   import java.util.Calendar;
10
11  public class MainActivity extends AppCompatActivity {
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17
18          TextView textView = findViewById(R.id.textView);
19          Button button = findViewById(R.id.button);
20          button.setOnClickListener(new View.OnClickListener() {
21              @Override
22              public void onClick(View v) {
23                  // Определяем объект Runnable
24                  Runnable runnable = new Runnable() {
25                      @Override
26                      public void run() {
27                          // получаем текущее время
28                          Calendar c = Calendar.getInstance();
29                          int hours = c.get(Calendar.HOUR_OF_DAY);
30                          int minutes = c.get(Calendar.MINUTE);
31                          int seconds = c.get(Calendar.SECOND);
32                          String time = hours + ":" + minutes + ":" + seconds;
33                          // отображаем в текстовом поле
34                          textView.setText(time);
35                      }
36                  };
37                  // Определяем объект Thread - новый поток
38                  Thread thread = new Thread(runnable);
39                  // Запускаем поток
40                  thread.start();
41              }
```

```
42            });
43        }
44    }
```

Отже, тут до кнопки прикріплено обробник натискання, який запускає новий потік. Створювати та запускати потік у Java можна різними способами. В даному випадку самі дії, які виконуються в потоці, визначаються методом **run()** об'єкта **Runnable** :

```
1    Runnable runnable = new Runnable() {
2        @Override
3        public void run() {
4            // получаем текущее время
5            Calendar c = Calendar.getInstance();
6            int hours = c.get(Calendar.HOUR_OF_DAY);
7            int minutes = c.get(Calendar.MINUTE);
8            int seconds = c.get(Calendar.SECOND);
9            String time = hours + ":" + minutes + ":" + seconds;
10           // отображаем в текстовом поле
11           textView.setText(time);
12       }
13   };
```

Наприклад, отримуємо поточний час і намагаємося відобразити його в елементі TextView.

Далі визначаємо об'єкт потоку - об'єкт **Thread** , який приймає об'єкт Runnable. І за допомогою методу **start()** запускаємо потік:

```
1    // Определяем объект Thread - новый поток
2    Thread thread = new Thread(runnable);
3    // Запускаем поток
4    thread.start();
```

Начебто нічого складного. Але якщо ми запустимо програму та натиснемо на кнопку, то ми зіткнемося з помилкою:

Оскільки змінювати стан візуальних елементів, звертатися до них ми можемо лише в основному потоці програми або UI потоку.

Для вирішення цієї проблеми взаємодії у вторинних потоках з елементами графічного інтерфейсу клас **View()** визначає метод **post()** :

```
1    boolean post (Runnable action)
```

Як параметр він приймає завдання, яке треба виконати, і повертає логічне значення - `true`, якщо завдання Runnable успішно поміщено в чергу повідомлення, або `false`якщо не вдалося розмістити в черзі

Також у класу View є аналогічний метод:

postDelayed() :

```
boolean postDelayed (Runnable action, long millsec)
```

Він також запускає завдання лише через певний проміжок часу в мілісекундах, який вказується в другому параметрі.

Так, змінимо код **MainActivity** наступним чином

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView textView = findViewById(R.id.textView);
        Button button = findViewById(R.id.button);
        button.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Определяем объект Runnable
                Runnable runnable = new Runnable() {
                    @Override
                    public void run() {
                        // получаем текущее время
                        Calendar c = Calendar.getInstance();
                        int hours = c.get(Calendar.HOUR_OF_DAY);
                        int minutes = c.get(Calendar.MINUTE);
                        int seconds = c.get(Calendar.SECOND);
                        String time = hours + ":" + minutes + ":" + seconds;
                        // отображаем в текстовом поле
                        textView.post(new Runnable() {
                            public void run() {
                                textView.setText(time);
                            }
                        });
                    }
```

```
                };
                // Определяем объект Thread - новый поток
                Thread thread = new Thread(runnable);
                // Запускаем поток
                thread.start();
            }
        });
    }
}
```

Тепер для оновлення TextView застосовується метод post:

```
textView.post(new Runnable() {
        public void run() {
            textView.setText(time);
    }
});
```

Тобто тут у методі об'єкта Runnable, `run()` що передається в метод, ми можемо звертатися до елементів візуального інтерфейсу і взаємодіяти з ними.`post()`

Подібним чином можна працювати з іншими віджетами, які успадковуються від класу **View** .

6.2 Приорітети потоків. Синхронізація потоків

У разі використання вторинних потоків слід враховувати наступний момент. Найбільш оптимальним способом є робота потоків з фрагментом, ніж безпосередньо з діяльністю. Наприклад, визначимо у файлі **activity_main.xml** наступний інтерфейс:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/statusView"
        app:layout_constraintLeft_toLeftOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/indicator"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/progressBtn" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/statusView"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Тут визначено кнопку для запуску вторинної задачі та елементи TextView та ProgressBar, які відображають індикацію виконання завдання.

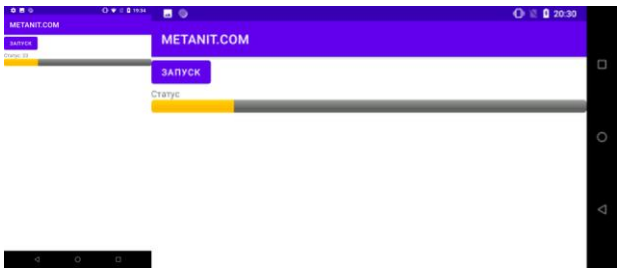
У класі **MainActivity** визначимо наступний код:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    int currentValue = 0;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ProgressBar indicatorBar = findViewById(R.id.indicator);
```

```
        TextView statusView = findViewById(R.id.statusView);
        Button btnFetch = findViewById(R.id.progressBtn);
        btnFetch.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Runnable runnable = new Runnable() {
                    @Override
                    public void run() {

                        for(; currentValue <= 100; currentValue++){
                            try {
                                statusView.post(new Runnable() {
                                    public void run() {
                                        indicatorBar.setProgress(currentValue);
                                        statusView.setText("Статус: " +
currentValue);
                                    }
                                });

                                Thread.sleep(400);
                            } catch (InterruptedException e) {
                                e.printStackTrace();
                            }
                        }
                    }
                };
                Thread thread = new Thread(runnable);
                thread.start();
            }
        });
    }
}
```

Тут за натисканням кнопки ми запускаємо завдання Runnable, в якому в циклі від 0 до 100 змінюємо показники ProgressBar та TextView, імітуючи деяку тривалу роботу.

Однак якщо в процесі роботи завдання ми змінимо орієнтацію мобільного пристрою, то відбудеться перестворення activity, і програма перестане працювати належним чином.



У разі проблема впирається у стан, яким оперує потік, саме - змінну `currentValue`, до значення якої прив'язані віджети в Activity.

6.2.1 Додавання ViewModel

Для подібних випадків як вирішення проблеми пропонується використовувати **ViewModel** . Отже, додамо до тієї ж папки, де знаходиться файл **MainActivity.java** , новий клас **MyViewModel** з наступним кодом:

```
1   package com.example.threadapp;
2
3   import androidx.lifecycle.LiveData;
4   import androidx.lifecycle.MutableLiveData;
5   import androidx.lifecycle.ViewModel;
6
7   public class MyViewModel  extends ViewModel  {
8
9       private MutableLiveData<Boolean> isStarted = new
10  MutableLiveData<Boolean>(false);
11      private MutableLiveData<Integer> value;
12      public LiveData<Integer> getValue() {
13          if (value == null) {
14              value = new MutableLiveData<Integer>(0);
15          }
16          return value;
17      }
18      public void execute(){
19
20          if(!isStarted.getValue()){
21              isStarted.postValue(true);
22              Runnable runnable = new Runnable() {
23                  @Override
24                  public void run() {
25
26                      for(int i = value.getValue();  i <= 100; i++){
27                          try {
28                              value.postValue(i);
29                              Thread.sleep(400);
30                          } catch (InterruptedException e) {
```

```
31                                  e.printStackTrace();
32                              }
33                          }
34                      }
35                  };
36                  Thread thread = new Thread(runnable);
37                  thread.start();
38              }
39          }
    }
```

Отже, тут визначено клас `MyViewModel`, який успадкований від класу **ViewModel** , спеціально призначеного для зберігання та керування станом чи моделлю.

Як стан тут визначено для об'єкта. Насамперед це числове значення, до яких будуть прив'язані віджети MainActivity. І по-друге, нам потрібен деякий індикатор того, що потік вже запущено, щоб за натисканням на кнопку не було запущено зайвих потоків.

Для зберігання числового значення призначена змінна `value`:

```
1   private MutableLiveData<Integer> value;
```

Для прив'язки до цього значення воно має тип `MutableLiveData`. А оскільки ми зберігатимемо в цій змінній числове значення, то тип змінної типізований типом `Integer`.

Для доступу ззовні класу до цього значення визначено метод `etValue`, який має тип `LiveData`і який при першому зверненні до змінної встановлює 0 або просто повертає значення змінної:

```
1   public LiveData<Integer> getValue() {
2       if (value == null) {
3           value = new MutableLiveData<Integer>(0);
4       }
5       return value;
6   }
```

Для індикації, чи потік запущено, визначена змінна `isStarted`, яка зберігає значення типу `Boolean`, тобто фактично `true` або `false`. За умовчанням вона має значення `false`(тобто потік не запущено).

Для зміни числового значення, до якого будуть прив'язані віджети, визначено метод `execute()`. Він запускає потік, якщо потік не запущено:

```
1   if(!isStarted.getValue()){
```

Далі перемикає значення змінної `isStarted` на `true`, оскільки ми запускаємо потік.

У потоці також запускається цикл

```
for(int i = value.getValue();  i <= 100; i++){
```

І в даному випадку ми користуємося перевагою класу ViewModel, що дозволяє автоматично зберігати свій стан.

Причому лічильник циклу як початкове значення бере значення зі змінної `value`і збільшується на одиницю, поки не дійде до ста.

У самому циклі змінюється значення змінної `value`за допомогою передачі значення метод`postValue()`

```
value.postValue(i);
```

Таким чином, у циклі здійсниться прохід від 0 до 100 і при кожній ітерації циклу буде змінюватися значення змінної value.

Тепер задіємо наш клас `MyViewModel` і для цього змінимо код класу **MainActivity** :

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.lifecycle.ViewModelProvider;

import android.os.Bundle;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ProgressBar indicatorBar = findViewById(R.id.indicator);
        TextView statusView = findViewById(R.id.statusView);
        Button btnFetch = findViewById(R.id.progressBtn);
        MyViewModel model = new ViewModelProvider(this).get(MyViewModel.class);

        model.getValue().observe(this, value -> {
            indicatorBar.setProgress(value);
            statusView.setText("Статус: " + value);
        });
        btnFetch.setOnClickListener(v -> model.execute());
    }
}
```

Щоб задіяти MyViewModel, створюємо об'єкт класу **ViewModelProvider** , конструктор якого передається об'єкт-власник ViewModel. В даному випадку це поточний об'єкт MainActivity:

```
new ViewModelProvider(this)
```

І далі за допомогою методу `get()` створюємо об'єкт класу ViewModel, який використовуватиметься в об'єкті MainActivity.

```
1   MyViewModel model = new ViewModelProvider(this).get(MyViewModel.class);
```

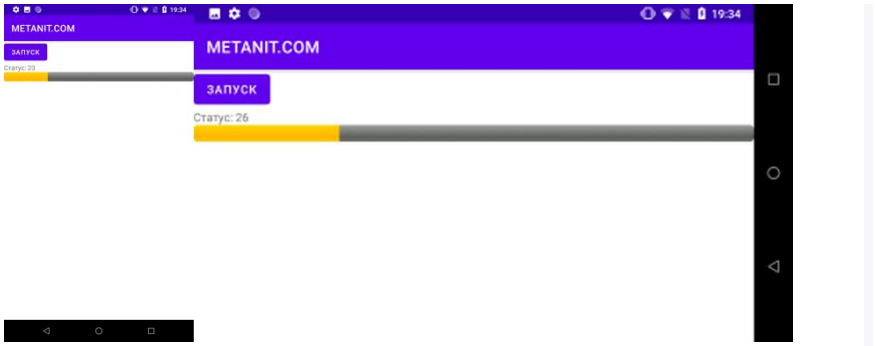
Отримавши об'єкт MyViewModel, визначаємо прослуховування змін його змінної `value` за допомогою методу **observe** :

```
1   model.getValue().observe(this, value -> {
2       indicatorBar.setProgress(value);
3       statusView.setText("Статус: " + value);
4   });
```

Метод `observe()` як перший параметр приймає власника функції обсервера - у разі поточний об'єкт MainActivity. А як другий параметр - функцію обсервера (а точніше об'єкт інтерфейсу `Observer`). Функція обсервера приймає один параметр - нове значення змінної, що відстежується (тобто в даному випадку змінної value). Отримавши нове значення змінної value, ми змінюємо параметри віджетів.

Таким чином, при кожній зміні значення змінної value віджети отримають її нове значення.

Тепер якщо ми запустимо програму, то незалежно від зміни орієнтації мийного пристрою фонове завдання продовжуватиме свою роботу:



Аналогічно ми можемо використовувати фрагменти. Отже, додамо до проекту новий фрагмент, який назвемо **ProgressFragment** .

Визначимо для нього новий файл розмітки інтерфейсу **fragment_progress.xml** :

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <androidx.constraintlayout.widget.ConstraintLayout
```

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/statusView"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent"/>

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toTopOf="@id/indicator"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@id/progressBtn" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/statusView"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Сам клас фрагмента **ProgressFragment** змінимо так:

```
package com.example.threadapp;

import android.os.Bundle;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProvider;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
```

```
import android.widget.ProgressBar;
import android.widget.TextView;


public class ProgressFragment extends Fragment {

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle save
{

        View view = inflater.inflate(R.layout.fragment_progress, container, false);

        ProgressBar indicatorBar = (ProgressBar) view.findViewById(R.id.indicator);
        TextView statusView = (TextView) view.findViewById(R.id.statusView);
        Button btnFetch = (Button)view.findViewById(R.id.progressBtn);

        MyViewModel model = new ViewModelProvider(requireActivity()).get(MyViewModel.c

        model.getValue().observe(getViewLifecycleOwner(), value -> {
            indicatorBar.setProgress(value);
            statusView.setText("Статус: " + value);
        });
        btnFetch.setOnClickListener(v -> model.execute());
        return view;
    }
}
```

Тут аналогічно застосовується клас MyViewModel. Єдиним чином для отримання асоційованої з фрагментом Activity тут застосовується метод **requireActivity()** . А для отримання власника життєвого циклу – метод `getViewLifecycleOwner`.

Тепер зв'яжемо фрагмент з діяльності. Для цього визначимо у файлі **activity_main.xml** наступний код:

```
<androidx.fragment.app.FragmentContainerView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/fragment_container_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:name="com.example.threadapp.ProgressFragment" />
```

А сам клас **MainActivity** скоротимо:

```
package com.example.threadapp;

import androidx.appcompat.app.AppCompatActivity;

```

```
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

І код із фрагементом працюватиме аналогічно.

6.3 Асинхронність

Щоб використовувати AsyncTask, нам потрібно:

1. Створити клас, похідний від AsyncTask (як правило, для цього створюється внутрішній клас в діяльності або у фрагменті)
2. Перевизначити один або кілька методів AsyncTask для виконання певної роботи у фоновому режимі
3. При необхідності створити об'єкт AsyncTask та викликати його метод `execute()` для початку роботи

Отже, створимо найпростіший додаток із використанням AsyncTask. Визначимо наступну розмітку інтерфейсу у файлі **activity_main.xml** :

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:orientation="vertical">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="22sp"
            android:id="@+id/clicksView"
            android:text="Clicks: 0"/>
        <Button
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/clicksBtn"
            android:text="Click" />
    </LinearLayout>

    <Button
        android:id="@+id/progressBtn"
        android:text="Запуск"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/statusView"
        android:text="Статус"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <ProgressBar
        android:id="@+id/indicator"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="0" />
</LinearLayout>
```

Тут визначено кнопку для запуску фонового потоку, а також текстове поле та прогрессбар для індикації виконання завдання. Крім того, тут визначені додаткова кнопка, яка збільшує число кліків, і текстове поле, яке виводить число кліків.

Далі визначимо в класі **MainActivity** наступний код:

```
import android.os.AsyncTask;
import android.os.SystemClock;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    int[] integers=null;
```

```
14      int clicks = 0;
15      ProgressBar indicatorBar;
16      TextView statusView;
17      TextView clicksView;
18      Button progressBtn;
19      Button clicksBtn;
20      @Override
21      protected void onCreate(Bundle savedInstanceState) {
22          super.onCreate(savedInstanceState);
23          setContentView(R.layout.activity_main);
24
25          integers = new int[100];
26          for(int i=0;i<100;i++) {
27              integers[i] = i + 1;
28          }
29          indicatorBar = (ProgressBar) findViewById(R.id.indicator);
30          statusView = findViewById(R.id.statusView);
31          progressBtn = findViewById(R.id.progressBtn);
32          progressBtn.setOnClickListener(new View.OnClickListener() {
33              @Override
34              public void onClick(View v) {
35
36                  new ProgressTask().execute();
37              }
38          });
39
40          clicksView = findViewById(R.id.clicksView);
41          clicksBtn = findViewById(R.id.clicksBtn);
42          clicksBtn.setOnClickListener(new View.OnClickListener() {
43              @Override
44              public void onClick(View v) {
45
46                  clicks++;
47                  clicksView.setText("Clicks: " + clicks);
48              }
49          });
50      }
51
52      class ProgressTask extends AsyncTask<Void, Integer, Void> {
53          @Override
54          protected Void doInBackground(Void... unused) {
55              for (int i = 0; i<integers.length;i++) {
56
57                  publishProgress(i);
58                  SystemClock.sleep(400);
```

```
59              }
60              return(null);
61          }
62          @Override
63          protected void onProgressUpdate(Integer... items) {
64              indicatorBar.setProgress(items[0]+1);
65              statusView.setText("Статус: " + String.valueOf(items[0]+1));
66          }
67          @Override
68          protected void onPostExecute(Void unused) {
69              Toast.makeText(getApplicationContext(), "Задача завершена",
70  Toast.LENGTH_SHORT)
71                      .show();
72          }
73      }
    }
```

Клас завдання ProgressTask визначено як внутрішній клас. Він успадковується не просто від AsyncTask, а від його типізованої версії `AsyncTask<Void, Integer, Void>`. Вона типізується трьома типами:

- Клас для зберігання інформації, яка потрібна для обробки задачі
- Тип об'єктів, що використовуються для індикації процесу виконання завдання
- Тип результату задачі

Ці типи можуть бути різними класами. В даному випадку сутність завдання полягатиме в переборі масиву integers, що представляє набір елементів Integer. І тут нам не треба передавати в завдання жодний об'єкт, тому перший тип іде як **Void** .

Для індикації перебору використовуються цілі числа, які показують, який об'єкт із масиву ми зараз перебираємо. Тому як другий тип використовується Integer.

Як третій тип використовується знову Void, оскільки в даному випадку не треба нічого повертати із завдання.

AsyncTask містить чотири методи, які можна перевизначити:

- Метод **doInBackground()** : виконується у фоновому потоці, повинен повертати певний результат
- Метод **onPreExecute()** : викликається з головного потоку перед запуском методу`doInBackground()`

- Метод **onPostExecute()** : виконується з головного потоку після завершення роботи методу`doInBackground()`
- Метод **onProgressUpdate()** : дозволяє сигналізувати користувачеві про виконання фонового потоку

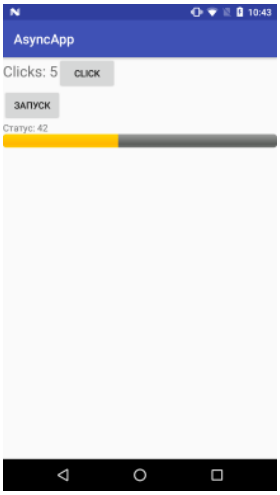
Так як метод doInBackground() не приймає нічого і не повертає нічого, то як його параметр використовується `Void...`- масив Void, і як тип, що повертається - теж Void. Ці типи відповідають першому і третьому типу в `AsyncTask<Void, Integer, Void>`.

Метод doInBackground() перебирає масив і за кожної ітерації повідомляє систему з допомогою методу **publishProgress(item)** . Так як у нашому випадку для індикації використовуються цілі числа, параметр `item`повинен представляти ціле число.

Метод `onProgressUpdate(Integer... items)`отримує передане вище число та застосовує його для налаштування текстового поля та прогресу.

Метод `onPostExecute()`виконується після завершення завдання і як параметр приймає об'єкт, який повертається методом doInBackground() - тобто в даному випадку об'єкт типу Void. Щоб сигналізувати про закінчення роботи, тут виводиться на екран спливаюче повідомлення.

Запустимо програму. Запустимо завдання, натиснувши кнопку:



При цьому, поки виконується завдання, ми можемо паралельно натискати на другу кнопку і збільшувати кількість кліків, або виконувати якусь іншу роботу в додатку.