# Programowanie urządzeń mobilnych

**Wielowątkowość**

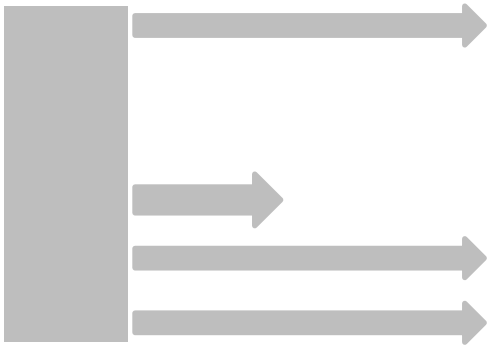
Tłumaczenie i adaptacja materiałów: dr Tomasz Xięski.
Na podstawie prezentacji udostępnionych przez Victor Matos, Cleveland State University.

Portions of this page are reproduced from work created and shared by Google and used according to terms

# Wielowątkowość

## Wątki

1. Wykorzystanie wątków przydaje się w sytuacjach, wykonywania więcej niż jednej czynności naraz np. ściąganie pliku z internetu. Jedno z rozwiązań tego problemu, które pozwala jednocześnie uruchamiać wiele akcji to obsługa ich w różnych **wątkach**.



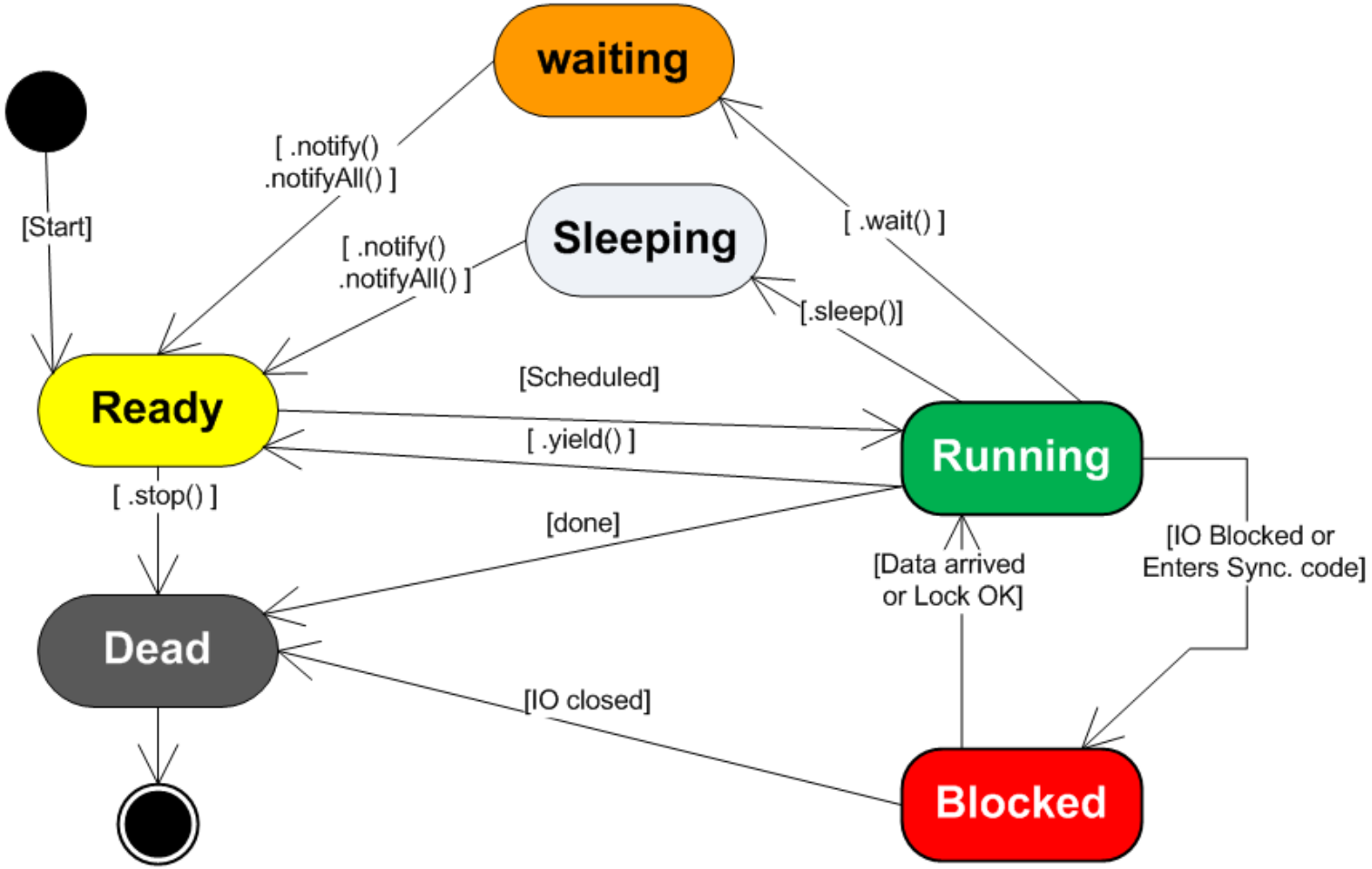
2. Maszyna wirtualna Javy zapewnia własną architekturę **wielowątkowości.**

3. Wątki w ramach tej samej maszyny wchodzą w interakcję wykorzystując **współdzielone obiekty** i **monitory**.

4. Każda maszyna wirtualna ma co najmniej jeden **wątek główny**.

5. Każdy wątek ma swój **stos wywołań**. Używany jest do wywołania metod, przekazywania parametrów, zarządzania zmiennymi lokalnymi itp..

# Cykl życia wątku w Javie



# Wielowątkowość

## Tworzenie i uruchamianie wątków

Istnieją 2 klasyczne strategie tworzenia i uruchamiania wątków:

**Opcja1** Stwórz nową instancję **Thread** przekazując jej obiekt **Runnable.**

```
Runnable myRunnable1 = new MyRunnableClass();
Thread t1 = new Thread(myRunnable1);
t1.start();
```

**Opcja2** Stwórz własną klasę, która rozszerza klasę **Thread** i redefiniuje metodę **run()**.

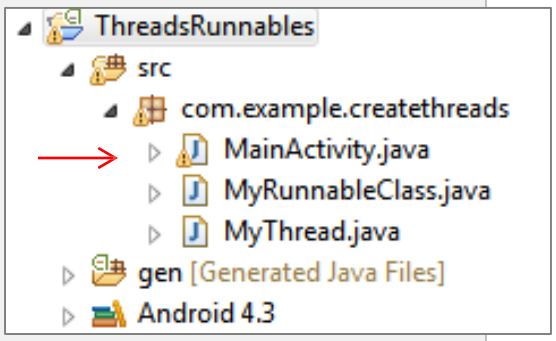
```
MyThread t2 = new MyThread();
t2.start();
```

W obu przypadkach, metoda **start()** musi być wywołana by uruchomić przetwarzanie w nowym wątku.

# Wielowątkowość

## Przykład1. Tworzenie dwóch wątków

```
public class MainActivity extends Activity {
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

1       Runnable myRunnable1 = new MyRunnableClass();
        Thread t1 = new Thread(myRunnable1);
        t1.start();

2       MyThread t2 = new MyThread();
        t2.start();

    }//onCreate
```




1. Wykorzystanie opcji 1.
2. Wykorzystanie opcji 2.

# Wielowątkowość

## Przykład1. Tworzenie dwóch wątków

```
1 →
public class MyRunnableClass implements Runnable {
   @Override
   public void run() {
      try {
         for (int i = 100; i < 105; i++){
2 →         Thread.sleep(1000);
            Log.e ("t1:<<runnable>>", "runnable talking: " + i);
         }
      } catch (InterruptedException e) {
          Log.e ("t1:<<runnable>>",  e.getMessage() );
      }

   }//run

}//class
```

1. Należy zaimplementować interfejs Runnable i nadpisać metodę run().

2. Wątek jest usypiany na sekundę.

# Wielowątkowość

## Przykład1. Tworzenie dwóch wątków

```
1 →
public class MyThread extends Thread{

   @Override
   public void run() {
      super.run();
      try {
         for(int i=0; i<5; i++){
2 →         Thread.sleep(1000);
            Log.e("t2:[thread]", "Thread talking: " + i);
         }
      } catch (InterruptedException e) {
          Log.e("t2:[thread]", e.getMessage() );
      }
   }//run

}//MyThread
```

1. Rozszerzanie klasy Thread.

2. Wątek jest usypiany na sekundę.

# Wielowątkowość

## Przykład1. Tworzenie dwóch wątków - testowanie

LogCat | Console | Problems | Search | Error Log | Lint Warnings

Saved Filters

no filters) (4458)

createthreads (Session Filt

Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.

| L... | T | P | TID | Application | Tag | Text |
|---|---|---|---|---|---|---|
| D | 1 | 1 | 1.. | com.example.createthreads | gralloc_goldfish | Emulator without GPU emu |
| E | 1 | 1 | 1.. | com.example.createthreads | t1:<<runnable>> | runnable talking: 100 |
| E | 1 | 1 | 1.. | com.example.createthreads | t2:[thread] | Thread talking: 0 |
| E | 1 | 1 | 1.. | com.example.createthreads | t1:<<runnable>> | runnable talking: 101 |
| E | 1 | 1 | 1.. | com.example.createthreads | t2:[thread] | Thread talking: 1 |
| E | 1 | 1 | 1.. | com.example.createthreads | t1:<<runnable>> | runnable talking: 102 |
| E | 1 | 1 | 1.. | com.example.createthreads | t2:[thread] | Thread talking: 2 |
| E | 1 | 1 | 1.. | com.example.createthreads | t1:<<runnable>> | runnable talking: 103 |
| E | 1 | 1 | 1.. | com.example.createthreads | t2:[thread] | Thread talking: 3 |
| E | 1 | 1 | 1.. | com.example.createthreads | t1:<<runnable>> | runnable talking: 104 |
| E | 1 | 1 | 1.. | com.example.createthreads | t2:[thread] | Thread talking: 4 |

**Przeplatane wykonywanie wątków t1 i t2.**

# Wielowątkowość

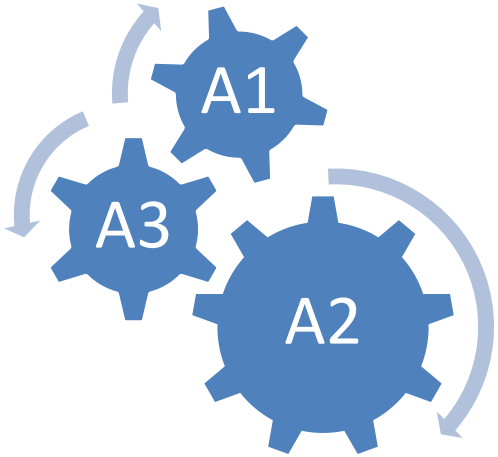
## Zalety wielowątkowości

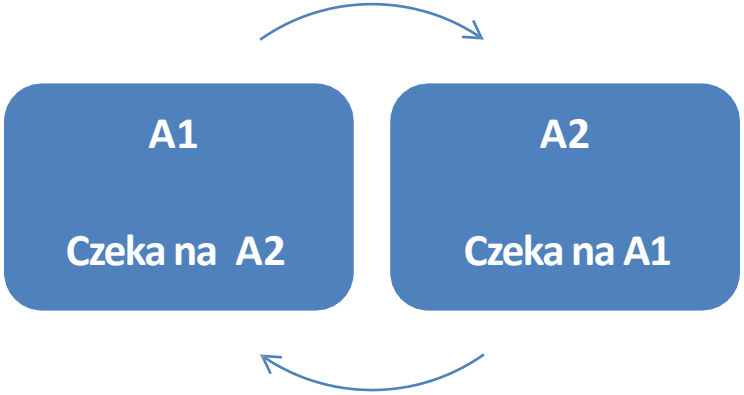
1. Różne składniki funkcjonalne aplikacji mogą być wyabstrahowane wokół pojęcia działań szeregowych lub równoległych.
2. Akcje szeregowe mogą być implementowane przy użyciu wspólnych metod klasy, podczas gdy działanie równoległe można przypisać do niezależnych wątków.
3. Wątki mogą udostępniać zasoby danych przechowywane w procesie, który wątki agreguje.
4. Tworzenie responsywnych aplikacji polega na umieszczeniu logiki odpowiedzialnej za kontrolę interakcji użytkownika z widokiem w głównym wątku aplikacji, a powolne akcje wykonywane są w tle.
5. Program wielowątkowy działa szybciej w systemach komputerowych, które mają wiele procesorów/rdzeni. Większość obecnych urządzeń z Androidem to konstrukcje wielordzeniowe.

# Wielowątkowość

## Wady wielowątkowości

1. Kod jest bardziej złożony.
2. Należy wykrywać/unikać **zakleszczenia.**



# Wielowątkowość

## Strategie dotyczące wielowątkowości w Androidzie

**Problem:** Aplikacja może wykonywać czasochłonne zadanie. Wówczas inne części aplikacji są blokowane.

**Cel:** Widok (oraz inne komponenty) powinien pozostawać responsywny pomimo wysokiego obciążenia.

**Rozwiązanie:** Android oferuje dwa rozwiązania:

1. Wykonuj kosztowne operacje w ***usłudze w tle***, wykorzystując *powiadomienia* do informowania użytkownika.
2. Wykonuj powolną operację w ***osobnym wątku***.

**Wykorzystanie wątków:** Interakcja pomiędzy wątkami w Androidzie (główny i poboczny) możliwa jest poprzez

(a) Obiekt klasy ***Handler*** w głównym wątku oraz

(b) Publikowanie obiektów ***Runnable*** do głównego wątku.

# Wielowątkowość

## Klasa Handler

- Główny wątek może używać tzw. kolejki ***MessageQueue*** by zarządzać interakcją między tworzonymi wątkami.

- Kolejka komunikatów działa jako kolejka priorytetów chroniona semaforami z możliwością przechwytywania informacji (tzw. tokenów) pochodzących z wątków pobocznych.

- Według protokołu wątki potomne muszą żądać pustych tokenów z kolejki przodka, wypełnić je, a następnie odesłać do kolejki rodzica.

- W architekturze Androida każdy wątek ma kolejkę MessageQueue. By z niej skorzystać, obiekt typu **Handler** musi zostać stworzony.

- Handler doda kolejną wiadomość (i obiekt *runnable*) do kolejki wiadomości rodzica (wraz z uwzględnieniem kolejności).

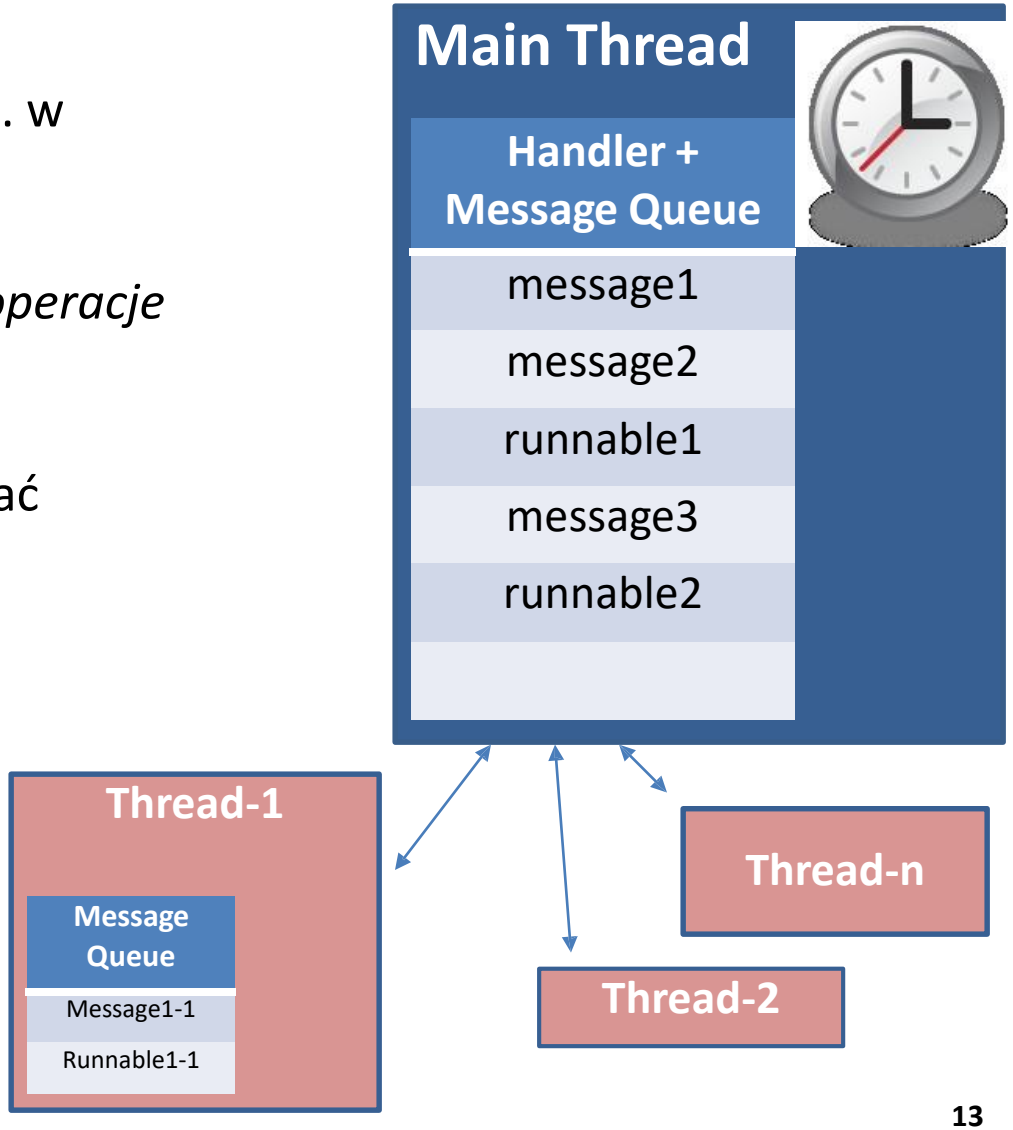
# Wielowątkowość

## Klasa Handler

**Handler** wykorzystywany jest m.in. w dwóch przypadkach:

(1) by zaplanować *wiadomości* i *operacje* wykonywane w przyszłości,

(2) by dodać akcję, która ma zostać wykonana w innym wątku.



# Wielowątkowość

## Wątki a widok

## Uwaga

**Wątki w tle nie mogą wchodzić w interakcję z interfejsem użytkownika**

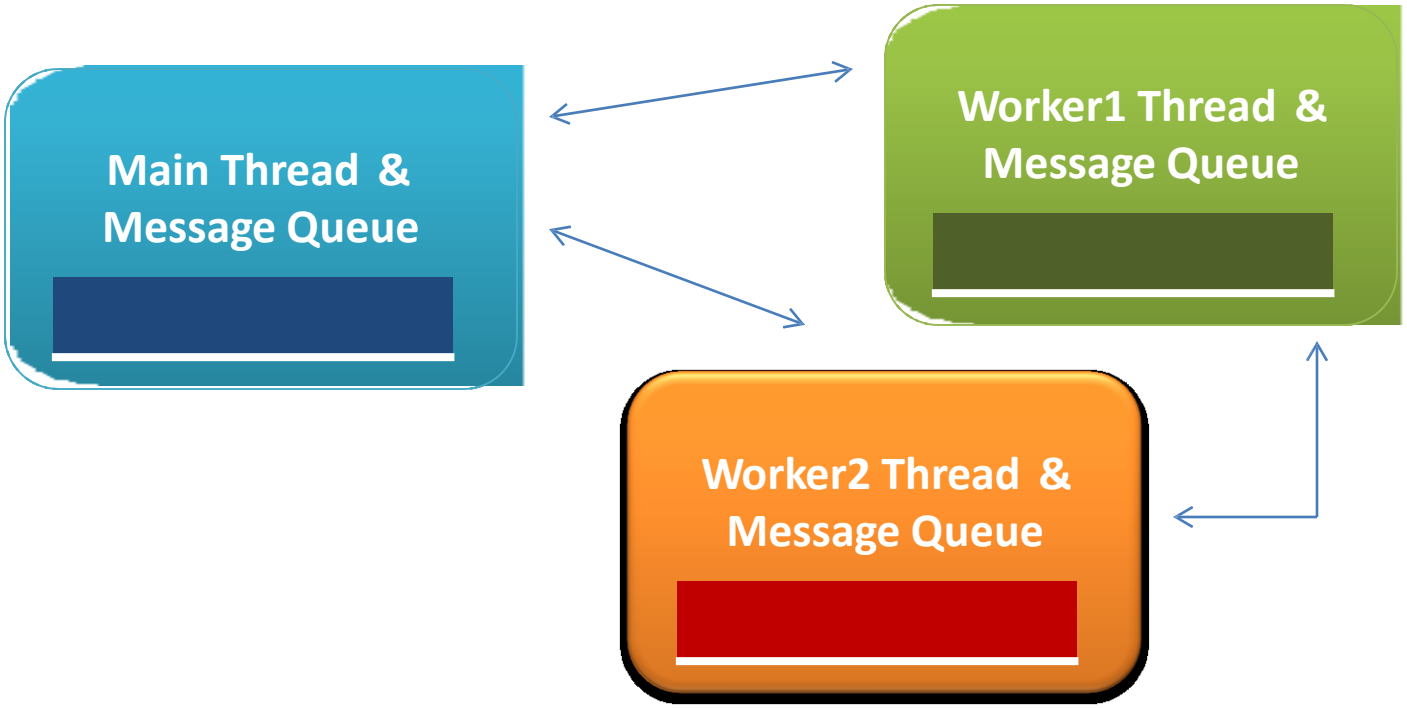
- Tylko główny wątek może uzyskać dostęp do widoku aktywności i interakcji z użytkownikiem.
- Prosty eksperyment. Dodaj wiadomość Toast do metod run() zaimplementowanych w przykładzie 1. Obie się nie wykonają!
- Publiczne pola klasy (zdefiniowane w głównym wątku) mogą być aktualizowane przez inne wątki.

# Wielowątkowość

## Komunikacja między wątkami

- Zwykle główny wątek ustawia handler by otrzymywać wiadomości z wątków pobocznych; jednakże *każdy wątek może zdefiniować swój własny handler*.

- Handler w wątku pobocznym tworzy lokalną kolejkę wiadomości, która może być używana by otrzymywać informacje z innych wątków pobocznych (oraz głównego).



# Wielowątkowość

## Protokół przekazywania wiadomości

1. Dany wątek, który chce komunikować się z głównym wątkiem, musi zarządzać **otrzymanie tokena** za pomocą metody ***obtainMessage*()**.

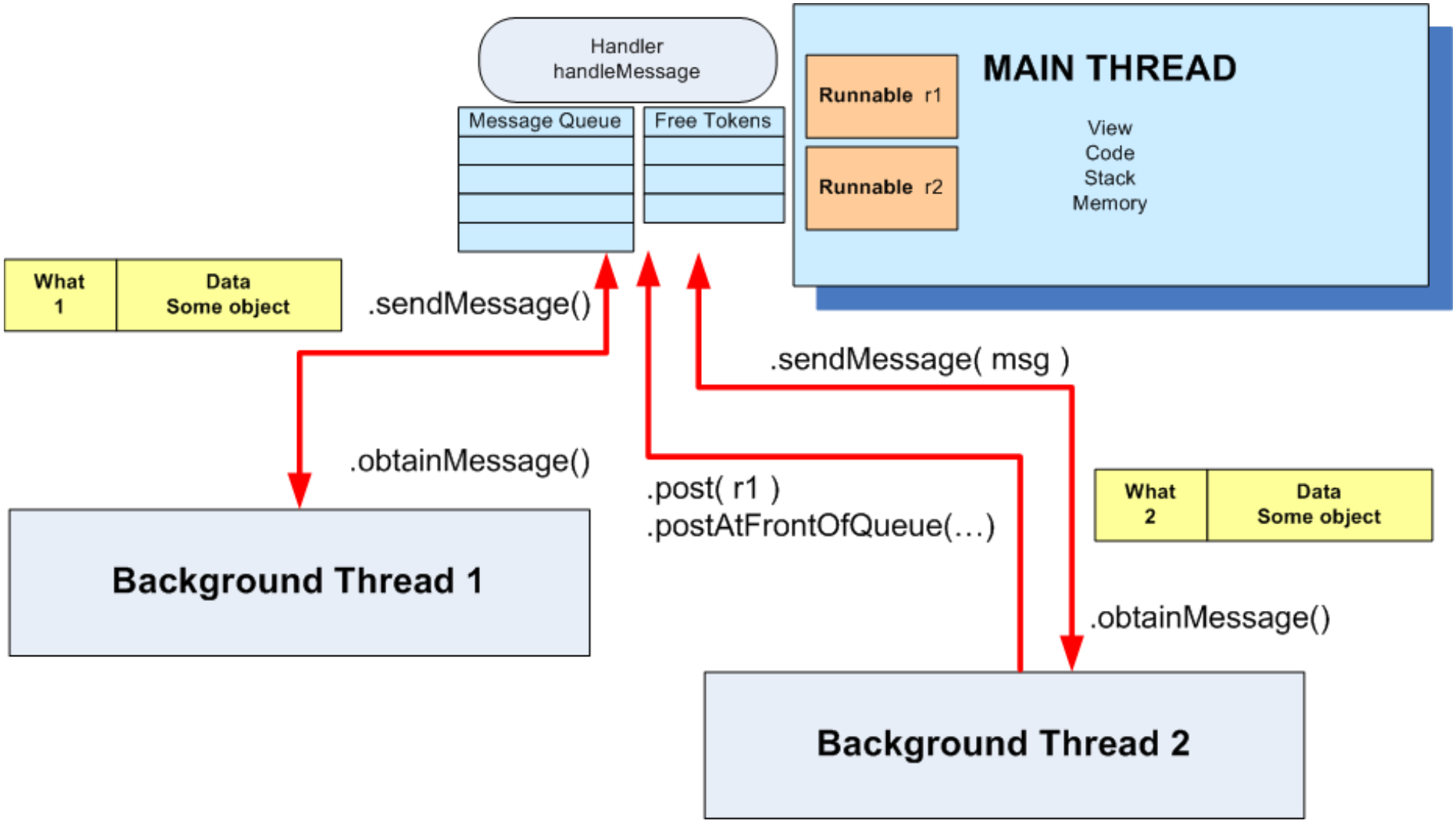
2. Po jego otrzymaniu, wątek może wprowadzić dane do **tokena** i dołączyć go do kolejki wiadomości **Handlera** wykorzystując metodę ***sendMessage*()**.

3. Handler wykorzystuje metodę ***handleMessage*()** do ciągłego przetwarzania wiadomości docierających do głównego wątku.

4. **Token** pobrany z kolejki wiadomości może
   - Zwrócić jakieś **dane** do głównego wątku,
   - Zażądać **wykonania** obiektu Runnable poprzez metodę ***post*()**.

# Wielowątkowość

## Protokół przekazywania wiadomości



# Wielowątkowość

## Handler. Wykorzystanie wiadomości do komunikacji

### Main Thread

```
...
Handler myHandler = new Handler() {

  @Override
  public void handleMessage(Message msg) {

     // do something with the message...
     // update GUI if needed!
     ...
  }//handleMessage

};//myHandler

...
```

### Background Thread

```
...
Thread backgJob = new Thread (new Runnable (){

   @Override
   public void run() {
    //...do some busy work here ...
    //get a token to be added to
     //the main's message queue
    Message msg = myHandler.obtainMessage();
      ...
     //deliver message to the
    //main's message-queue
    myHandler.sendMessage(msg);
   }//run

});//Thread

//this call executes the parallel thread
backgroundJob.start();
...
```

# Wielowątkowość

## Handler. Using Runnables to Communicate

### Main Thread

```
...
Handler      myHandler = new Handler();
@Override
public void onCreate(
          Bundle savedInstanceState) {
 ...
 Thread myThread1 =
        new Thread(backgroundTask,
                   "backAlias1");
 myThread1.start();

}//onCreate

...
//this is the foreground runnable
private Runnable foregroundTask
   = new Runnable() {
  @Override
  public void run() {
   // work on the UI if needed
}
...
```

### Background Thread

```
// this is the "Runnable" object
// representing the background thread

private Runnable backgroundTask
                  = new Runnable () {
  @Override
  public void run() {
     ... Do some background work here
    myHandler.post(foregroundTask);

  }//run

};//backgroundTask
```

# Wielowątkowość

## Handler: metoda obtainMessage

By wysłać wiadomość do handlera, wątek musi najpierw wywołać obtainMessage() by wyłuskać stosowny obiekt z kolejki.

Istnieje wiele wersji metody **obtainMessage().** Umożliwiają one stworzenie pustego obiektu wiadomości bądź takiego zawierającego dodatkowe parametry.

**Przykład**

```
//  assume thread 1 produces some local data
String localData = "Greetings from thread 1";

//  thread 1 requests a message & adds localData to it
Message mgs = myHandler.obtainMessage (1, localData);
```

# Wielowątkowość

## Handler: metoda sendMessage

Istnieje wiele wersji metod **sendMessage...()**, które mogą być wykrozytane przez wątki poboczne do wysłania wiadomości do innego (głównego) wątku.

- **sendMessage()** dodaje wiadomość do końca kolejki.

- **sendMessageAtFrontOfQueue()** dodaje wiadomość na sam początek kolejki, zatem przejmuje ona najwyższy priorytet i znajduje się przed wszystkimi innymi.

- **sendMessageAtTime()** dodaje wiadomość do kolejki o zadanym czasie, wyrażonej w formie milisekund względem czasu działania systemu ( SystemClock.uptimeMillis() )

- **sendMessageDelayed()** dodaje wiadomość do kolejki po określonym opóźnieniu, wyrażonym w milisekundach.

# Wielowątkowość

## Handler: przetwarzanie wiadomości

By przetworzyć wiadomości przesyłane przez inne wątki, dany Handler musi implementować nasłuchiwacz:

```
handleMessage( Message msg )
```

Który zostanie wywołany wraz z każdorazowym pojawianiem się wiadomości w kolejce.

Wówczas, handler może np. zaktualizować widok. Jednakże należy to wykonać szybko, bo wszystkie inne operacje na widoku czekają na zakończenie działania handlera.

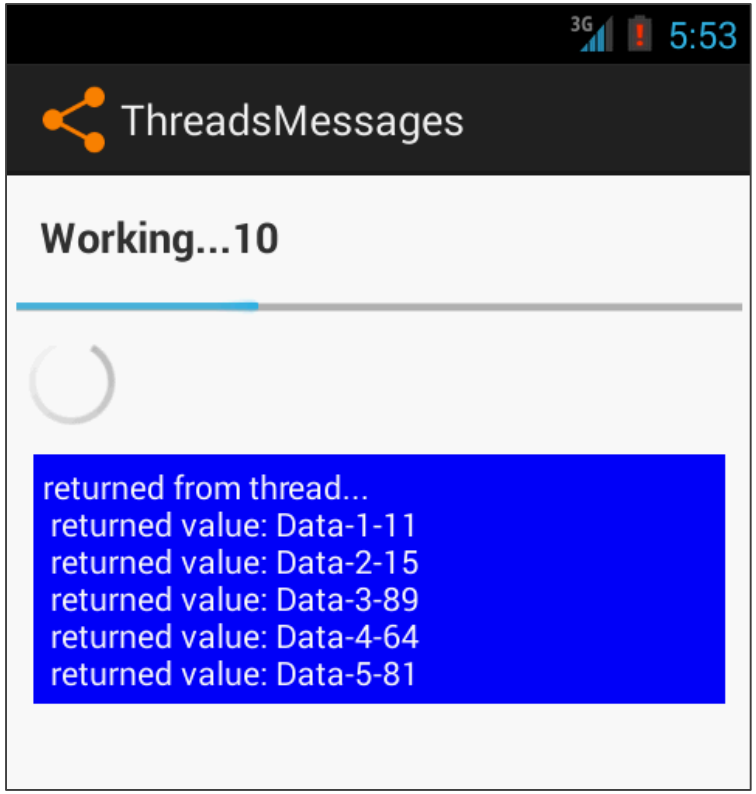
# Wielowątkowość

## Przykład 2. Komunikacja między wątkami

W tym przykładzie główny wątek przedstawia poziomy i okrągły pasek postępu sygnalizując postępy pracy wątku pobocznego.

Aby zasymulować pracę wykonywaną przez Wątek podobny, losowo wygenerowany wynik będzie okresowo wysyłany do głównego wątku.

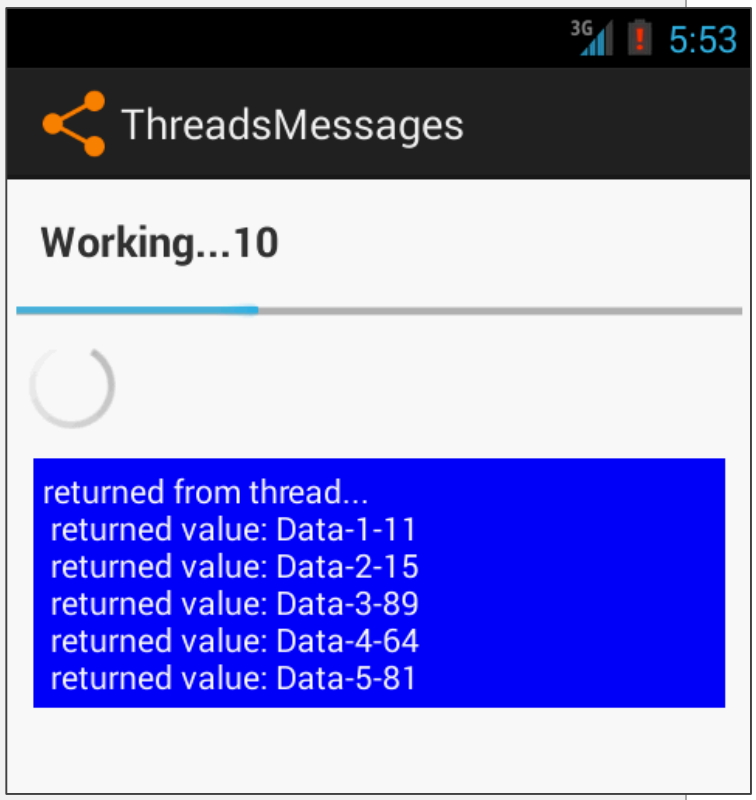
Na podstawie dostępnych komunikatów nastąpi aktualizacja widoku (pasków postępu).



# Wielowątkowość

## Przykład 2. Komunikacja między wątkami – układ XML

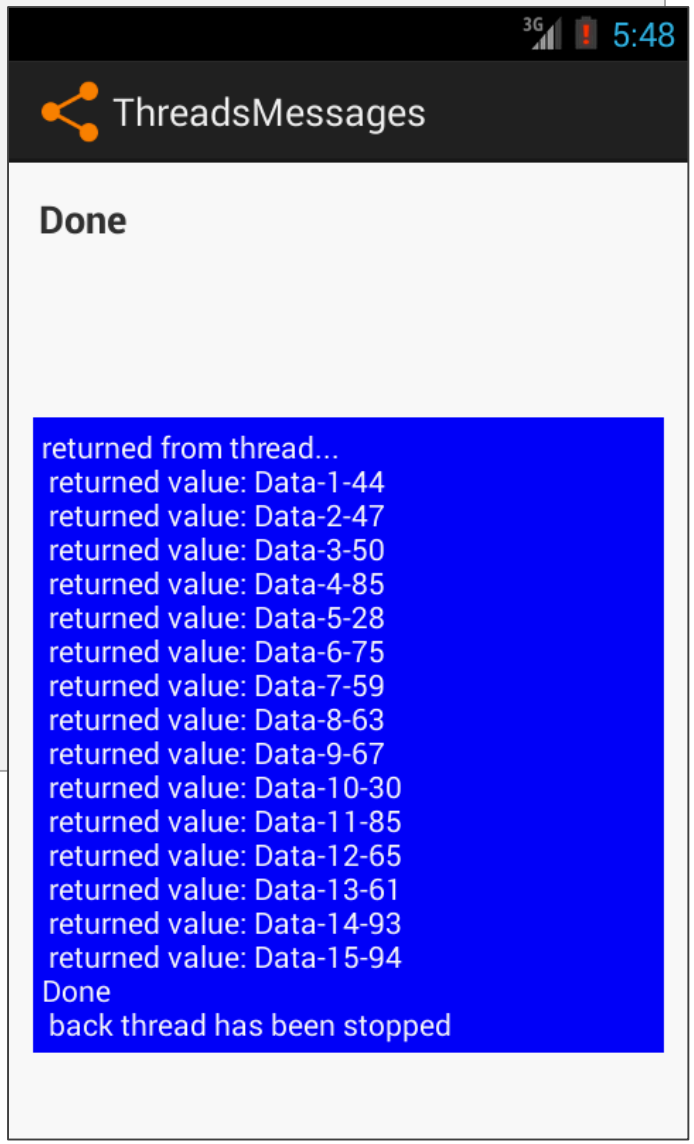
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#44ffff00"
    android:orientation="vertical"
    android:padding="4dp" >

    <TextView
        android:id="@+id/txtWorkProgress"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="10dp"
        android:text="Working ...."
        android:textSize="18sp"
        android:textStyle="bold" />

    <ProgressBar android:id="@+id/progress1"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <ProgressBar

        android:id="@+id/progress2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```



# Wielowątkowość

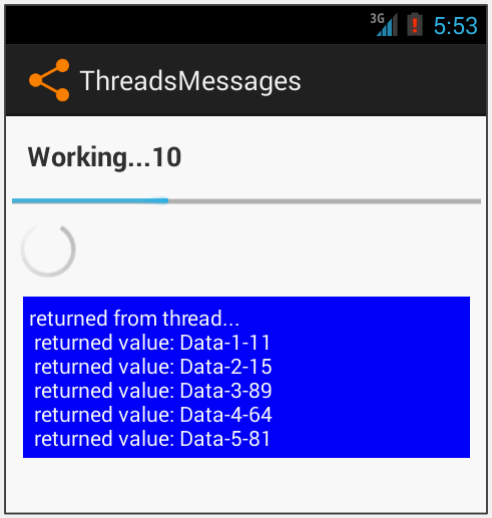
## Przykład 2. Komunikacja między wątkami – układ XML

```
    <ScrollView android:id="@+id/myscroller"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >

        <TextView
            android:id="@+id/txtReturnedValues"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="7dp"
            android:background="#ff0000ff"
            android:padding="4dp "
            android:text="returned from thread..."
            android:textColor="@android:color/white"
            android:textSize="14sp" />
    </ScrollView>

</LinearLayout>
```



# Wielowątkowość

## Przykład 2. Komunikacja między wątkami - MainActivity.java




```java
public class ThreadsMessages extends Activity {

   ProgressBar bar1;
   ProgressBar bar2;

   TextView msgWorking;
   TextView msgReturned;
   ScrollView myScrollView;


   // this is a control var used by backg. threads
   boolean isRunning = false;

   // lifetime (in seconds) for background thread
   final int MAX_SEC = 30;

   //String globalStrTest = "global value seen by all threads ";
   int globalIntTest = 0;
```

# Wielowątkowość

## Przykład 2. Komunikacja między wątkami - MainActivity.java

```
Handler handler = new Handler() {
   @Override
   public void handleMessage(Message msg) {
      String returnedValue = (String)msg.obj;
      //do something with the value sent by the background thread
      here msgReturned.append("\n returned value: " + returnedValue
      ); myScrollView.fullScroll(View.FOCUS_DOWN);
      bar1.incrementProgressBy(1);
      //testing early termination
      if (bar1.getProgress() == MAX_SEC){
         msgReturned.append(" \nDone \n back thread has been stopped");
         isRunning = false;
      }

      if (bar1.getProgress() == bar1.getMax()){
         msgWorking.setText("Done");
         bar1.setVisibility(View.INVISIBLE);
         bar2.setVisibility(View.INVISIBLE);
      }
      else {
         msgWorking.setText("Working..." + bar1.getProgress() );
      }
   }
}; //handler
```

# Wielowątkowość

## Przykład 2. Komunikacja między wątkami - MainActivity.java

```
@Override
public void onCreate(Bundle icicle) {
   super.onCreate(icicle);
   setContentView(R.layout.main);

   bar1 = (ProgressBar) findViewById(R.id.progress1);
5  bar1.setProgress(0);
   bar1.setMax(MAX_SEC);

   bar2 = (ProgressBar) findViewById(R.id.progress2);

   msgWorking = (TextView)findViewById(R.id.txtWorkProgress);
   msgReturned = (TextView)findViewById(R.id.txtReturnedValues);

   //globalStrTest += "XXX"; // slightly change the global string
   globalIntTest = 1;

}//onCreate
```

# Wielowątkowość

## Przykład 2. Komunikacja między wątkami - MainActivity.java

```
   public void onStart() {
      super.onStart();
      // this code creates the background activity where busy work is done
      Thread background = new Thread(new Runnable() {
         public void run() {
            try {
               for (int i = 0; i < MAX_SEC && isRunning; i++) {
                  //try a Toast method here (it will not work!)
                  //fake busy busy work here
6 →               Thread.sleep(1000);  // 1000 msec.

                  // this is a locally generated value between 0-100
                  Random rnd = new Random();
                  int localData = (int) rnd.nextInt(101);
                  //we can see and change (global) class variables
                  String data = "Data-" + globalIntTest + "-" + localData;
                  globalIntTest++;
                  //request a message token and put some data in it
7 →               Message msg = handler.obtainMessage(1, (String)data);

                  // if this thread is still alive send the
                  message if (isRunning) {
8 →                  handler.sendMessage(msg);
                  }
               }
            }
```

# Wielowątkowość

## Przykład 2. Using Handler & Messages - MainActivity.java

```
 catch (Throwable t) {
            // just end the background thread
            isRunning = false;
            }
         }
      });// Tread

      isRunning = true;
      background.start();

   }//onStart


   public void onStop() {
      super.onStop();
      isRunning = false;
   }//onStop
}//class
```

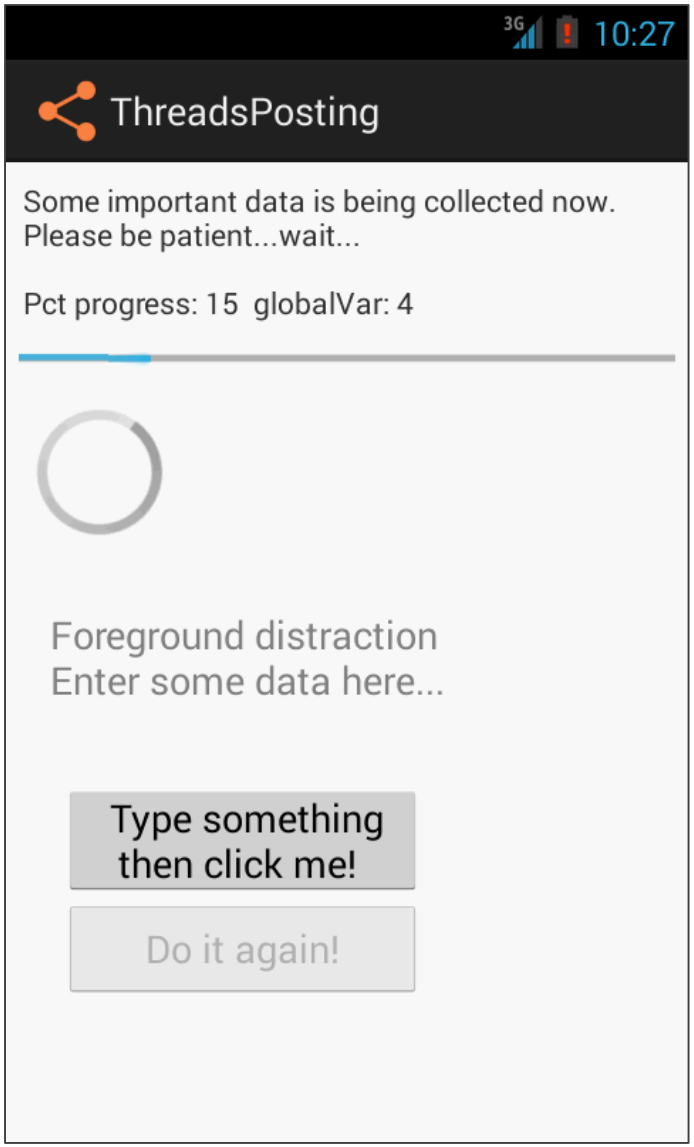
9

# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable

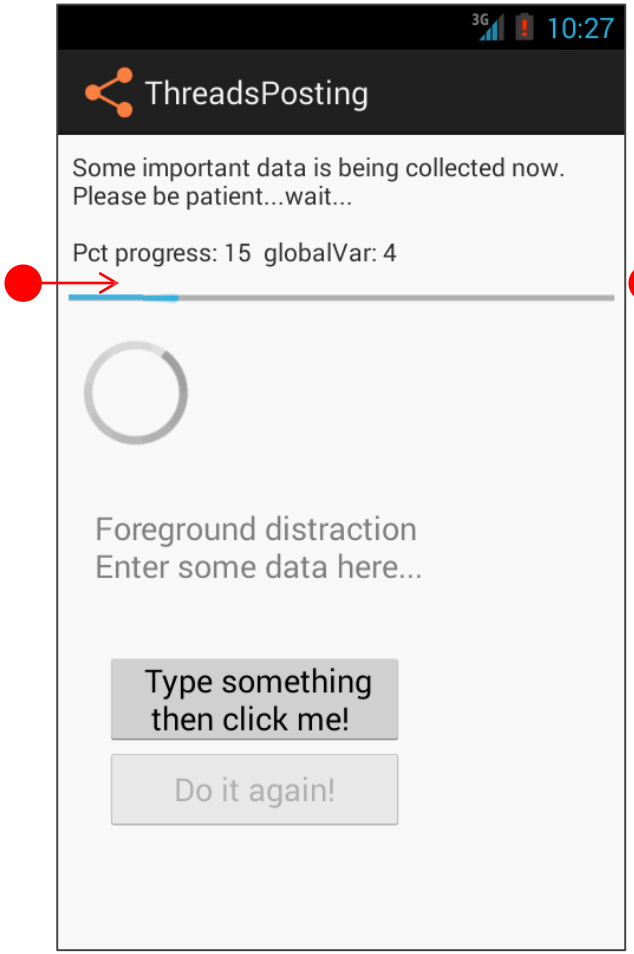
Przykład bazuje na założeniach z poprzedniego.

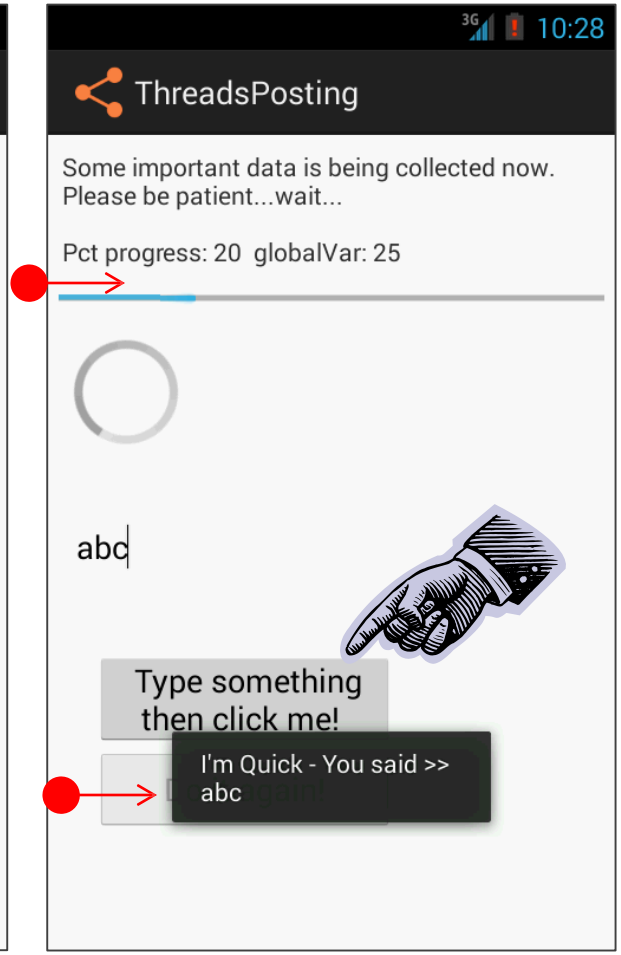
Tym razem zostanie wykorzystana metoda Post() i obiekty Runnable by zasymulować działania użytkownika z głównego wątku.

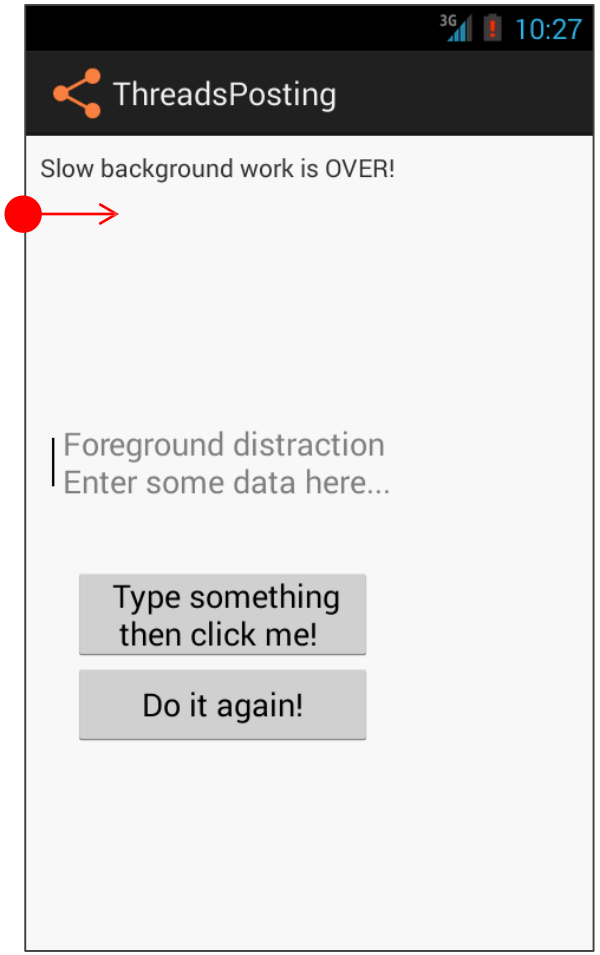


# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable



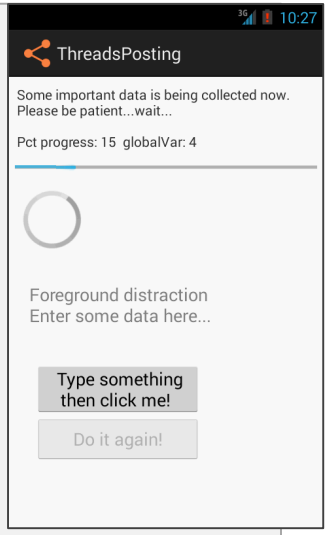



# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – układ XML

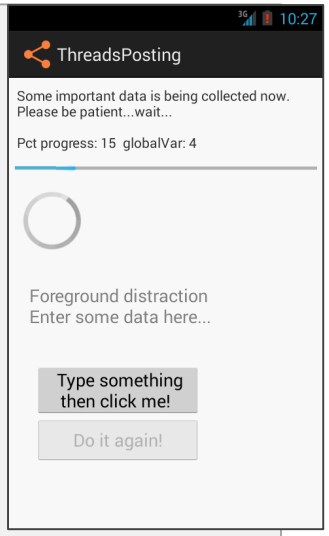
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#22002222"
    android:orientation="vertical"
    android:padding="6dp" >

    <TextView
        android:id="@+id/lblTopCaption"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="2dp"
        android:text=
            "Some important data is been collected now. Patience please..." />

    <ProgressBar android:id="@+id/myBarHor"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="30dp" />

    <ProgressBar android:id="@+id/myBarCir"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```



# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – układ XML

```
    <EditText android:id="@+id/txtBox1"
        android:layout_width="match_parent"
        android:layout_height="78dp"
        android:layout_margin="10dp"
        android:background="#ffffffff"
        android:textSize="18sp" />

    <Button
        android:id="@+id/btnDoSomething"
        android:layout_width="170dp"

        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:padding="4dp"
        android:text=" Type Something Then click me! " />

    <Button android:id="@+id/btnDoItAgain"
        android:layout_width="170dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:padding="4dp"
        android:text=" Do it Again! " />

</LinearLayout>
```



# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – MainActivity

```
public class MainActivity extends Activity {
   ProgressBar myBarHorizontal;
   ProgressBar myBarCircular;

   TextView lblTopCaption;
   EditText txtDataBox;
   Button btnDoSomething;
   Button btnDoItAgain;
   int progressStep = 5;
   final int MAX_PROGRESS = 100;

   int globalVar = 0;
   int accum = 0;

   long startingMills = System.currentTimeMillis();
   boolean isRunning = false;
   String PATIENCE = "Some important data is being collected now. "
         + "\nPlease be patient...wait...\n ";

1  Handler myHandler = new Handler();

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);

      lblTopCaption = (TextView) findViewById(R.id.lblTopCaption);
```

# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – MainActivity

```
    myBarHorizontal = (ProgressBar) findViewById(R.id.myBarHor);
    myBarCircular = (ProgressBar) findViewById(R.id.myBarCir);

    txtDataBox = (EditText) findViewById(R.id.txtBox1);
    txtDataBox.setHint(" Foreground distraction\n Enter some data here...");

    btnDoItAgain = (Button) findViewById(R.id.btnDoItAgain);
    btnDoItAgain.setOnClickListener(new OnClickListener() {
       @Override
       public void onClick(View v) {
          onStart();
       }// onClick
    });// setOnClickListener

    btnDoSomething = (Button) findViewById(R.id.btnDoSomething);
    btnDoSomething.setOnClickListener(new OnClickListener() {
       @Override
       public void onClick(View v) {
          String text = txtDataBox.getText().toString();
❷ →       Toast.makeText(MainActivity.this, "I'm quick - You said >>
          \n"
                        + text, 1).show();
       }// onClick
    });// setOnClickListener

 }// onCreate
```

# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – MainActivity

```
@Override
protected void onStart() {
   super.onStart();
   // prepare UI components
   txtDataBox.setText("");
   btnDoItAgain.setEnabled(false);

   // reset and show progress bars accum = 0;
   myBarHorizontal.setMax(MAX_PROGRESS);
   myBarHorizontal.setProgress(0);
   myBarHorizontal.setVisibility(View.VISIBLE)
   ; myBarCircular.setVisibility(View.VISIBLE);

   // create-start background thread were the busy work will be done
   Thread myBackgroundThread = new Thread( backgroundTask, "backAlias1" );
   myBackgroundThread.start();

}
```

3

# Wielowątkowość

## Przykład 3. Wykorzystanie obiektów Runnable – MainActivity

```
// FOREGROUND
// this foreground Runnable works on behave of the background thread,
// its mission is to update the main UI which is unreachable to back worker
private Runnable foregroundRunnable = new Runnable() {
   @Override
4  public void run() {
      try {
         // update UI, observe globalVar is changed in back thread
         lblTopCaption.setText( PATIENCE
                     + "\nPct progress: " + accum
                     + "  globalVar: " + globalVar );
         // advance ProgressBar
         myBarHorizontal.incrementProgressBy(progressStep);
         accum += progressStep;

         // are we done yet?
         if (accum >= myBarHorizontal.getMax()) {
            lblTopCaption.setText("Slow background work is OVER!");
            myBarHorizontal.setVisibility(View.INVISIBLE);
            myBarCircular.setVisibility(View.INVISIBLE);
            btnDoItAgain.setEnabled(true);
         }
      } catch (Exception e) {
         Log.e("<<foregroundTask>>", e.getMessage());
      }
   }
}; // foregroundTask
```

**Definicja nowego Runnable w ramach głównego wątku.**

**Zostanie on uruchomiony później**

# Wielowątkowość

## Example 3. Handler & Post & Runnables - MainActivity

```
   // BACKGROUND
   // this is the back runnable that executes the slow work

   private Runnable backgroundTask = new Runnable() {
      @Override
      public void run() {
         // busy work goes here...
         try {
            for (int n = 0; n < 20; n++) {
               // this simulates 1 sec. of busy activity
               Thread.sleep(1000);
               // change a global variable here...
               globalVar++;
               // try: next two UI operations should NOT work
               // Toast.makeText(getApplication(), "Hi ", 1).show();
               // txtDataBox.setText("Hi ");

               // wake up foregroundRunnable delegate to speak for you
               myHandler.post(foregroundRunnable);
            }
         } catch (InterruptedException e) {
            Log.e("<<foregroundTask>>", e.getMessage());
         }
      }// run
   };// backgroundTask

}// ThreadsPosting
```

5 →

6 →

**Uruchomienie uprzednio zdefiniowanego Runnable**

# Wielowątkowość

## Wykorzystanie klasy AsyncTask

```
AsyncTask <Params, Progress, Result>
```

| Parametry generyczne | |
|---|---|
| **Params**: | Typ danych wejściowych wysyłanych do wątku przed rozpoczęciem. |
| **Progress**: | Typ danych w jakich reprezentowany jest postęp wykonywania działania wątku. |
| **Result**: | Typ rezultatu wykonywanych działań w tle. |

Jeżeli któraś sekcja nie jest używana należy wykorzystać **Void**

# Wielowątkowość

## Wykorzystanie klasy AsyncTask

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {

    // Begin - can use UI thread here
    protected void onPreExecute() {

    }

    // this is the SLOW background thread taking care of heavy tasks
    // cannot directly change UI
    protected Void doInBackground(final String... args) {
    ... publishProgress((Long) someLongValue);
    }

    // periodic updates - it is OK to change UI
    @Override
    protected void onProgressUpdate(Long... value) {

    }

    // End - can use UI thread here
    protected void onPostExecute(final Void unused) {

    }
}
```

1 2 3 4

# Wielowątkowość

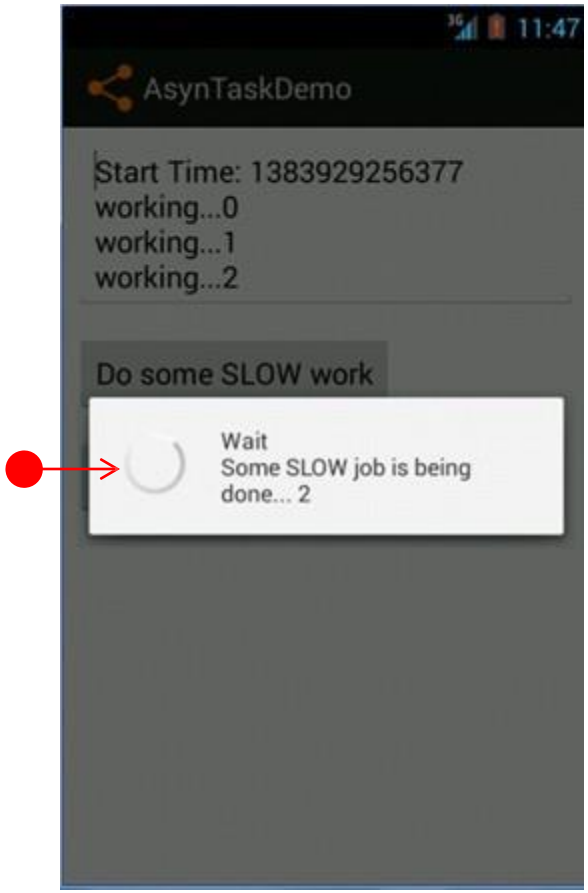
## Wykorzystanie klasy AsyncTask

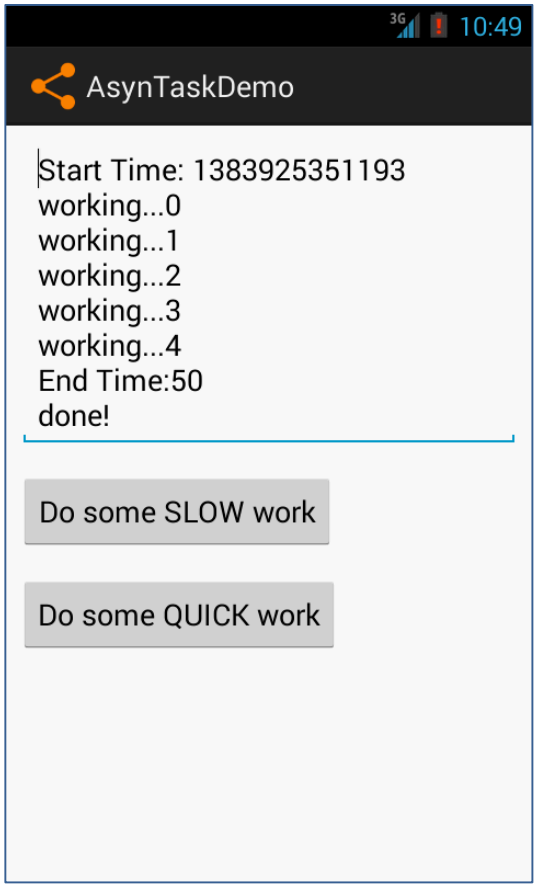
| Metody |
| --- |
| **onPreExecute(),** wywoływana w wątku interfejsu użytkownika zaraz po uruchomieniu wątku pobocznego. Ten krok jest zwykle używany do konfigurowania zadania, na przykład poprzez wyświetlanie paska postępu w interfejsie użytkownika. |
| **doInBackground(Params...),** uruchamiana jest na rzecz nowego wątku zaraz po tym jak metoda *onPreExecute*() zakończy działanie. Tutaj definiowane są długotrwałe operacje wykonywane w nowym wątku. Krok ten może również wykorzystywać metodę *publishProgress(Progress...)* by zaktualizować progres wykonania przetwarzania. Wartości te najczęściej służą do aktualizacji widoku na wątku głównym, za pośrednictwem metody *onProgressUpdate(Progress...)*. |
| **onProgressUpdate(Progress...)**, wywoływana w wątku interfejsu użytkownika zaraz po metodzie *publishProgress(Progress...)*. Ta metoda służy do informowaniu o skali postępu dokonanej przez wątek, który dalej pracuje w tle. |
| **onPostExecute(Result)**, wywoływane w wątku interfejsu użytkownika po zakończeniu przetwarzania w tle. Wynik przetwarzania w tle jest przekazywany do tej metody jako parametr. |

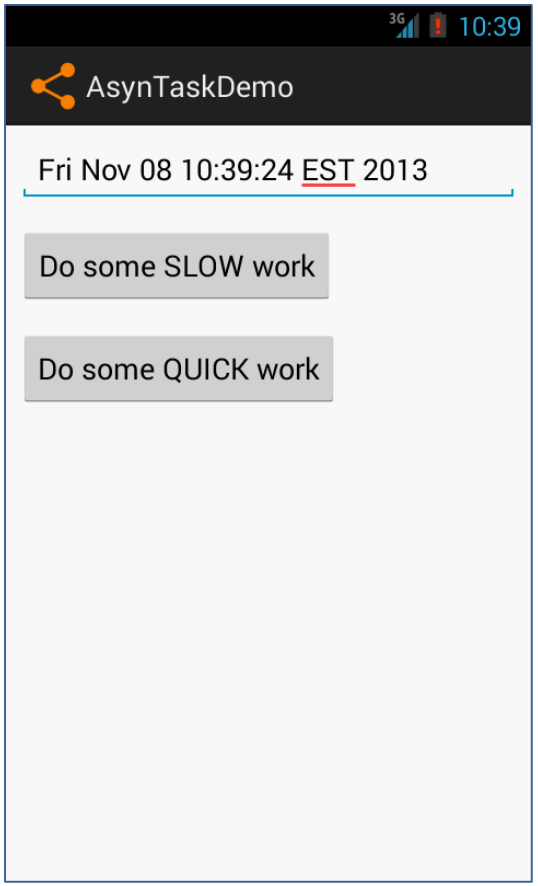
Źródło: http://developer.android.com/reference/android/os/AsyncTask.html

# Wielowątkowość

## Przykład 4: Wykorzystanie klasy AsyncTask



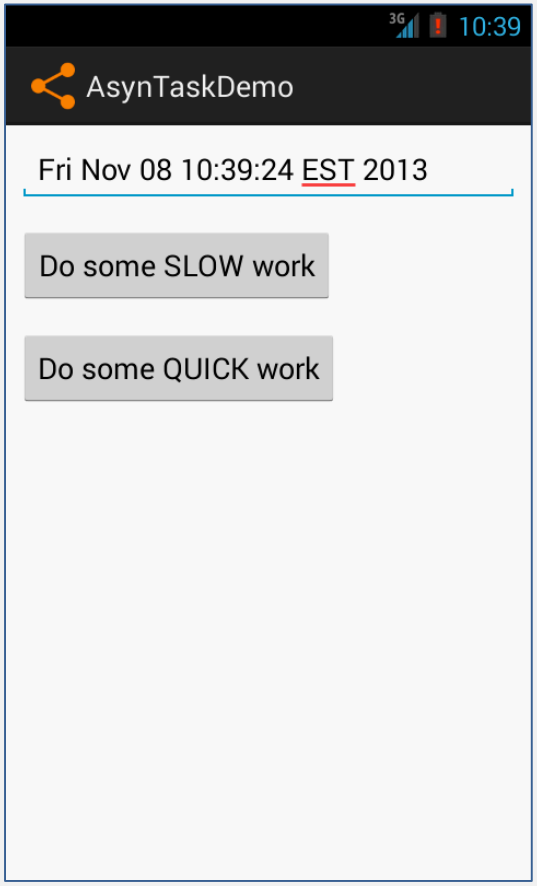




Główny wątek wywołuje instancję **AsyncTask** by wykonać długotrwałą operację. Metoda **doInBackgroud(…)** wykonuje przetwarzanie oraz cyklicznie wykorzystuje funkcję **onProgressUpdate(…)** by odświeżyć główny widok. W tym przykładnie dodawane są linie do pola tekstowego głównego wątku oraz prezentowany jest dialog typu **ProgressDialog**.

# Wielowątkowość

## Przykład 4: Wykorzystanie klasy AsyncTask – układ XML

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

    <EditText android:id="@+id/txtMsg"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="7dp" />

    <Button android:id="@+id/btnSlow"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some SLOW work" />

    <Button android:id="@+id/btnQuick"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="7dp"
        android:text="Do some QUICK work" />

</LinearLayout>
```



# Wielowątkowość

## Przykład 4: Wykorzystanie klasy AsyncTask – MainActivity

```
public class MainActivity extends Activity {
   Button btnSlowWork;
   Button btnQuickWork;
   EditText txtMsg;
   Long startingMillis;

   @Override
   public void onCreate(Bundle savedInstanceState)
      { super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      txtMsg = (EditText) findViewById(R.id.txtMsg);

      // slow work...for example: delete databases: “dummy1” and
      “dummy2” btnSlowWork = (Button) findViewById(R.id.btnSlow);
      this.btnSlowWork.setOnClickListener(new OnClickListener() {
         public void onClick(final View v) {
            new VerySlowTask().execute("dummy1", "dummy2");
         }
      });

      btnQuickWork = (Button) findViewById(R.id.btnQuick);
      this.btnQuickWork.setOnClickListener(new OnClickListener() {
         public void onClick(final View v) {
            txtMsg.setText((new Date()).toString()); // quickly show today’s date
         }
      });

   }// onCreate
```

1

# Wielowątkowość

## Przykład 4: Wykorzystanie klasy AsyncTask – MainActivity

```
private class VerySlowTask extends AsyncTask<String, Long, Void> {
   private final ProgressDialog dialog = new
   ProgressDialog(MainActivity.this); String waitMsg = "Wait\nSome SLOW job is
   being done... ";
❷ →
   protected void onPreExecute() {
      startingMillis = System.currentTimeMillis();
      txtMsg.setText("Start Time: " +
      startingMillis);
      this.dialog.setMessage(waitMsg);
      this.dialog.setCancelable(false); //outside touch doesn't dismiss
❸ →   you this.dialog.show();
   }

   protected Void doInBackground(final String... args) {
      // show on Log.e the supplied dummy arguments
      Log.e("doInBackground>>", "Total args: " + args.length
      ); Log.e("doInBackground>>", "args[0] = " + args[0] );

      try {
         for (Long i = 0L; i < 5L; i++) {
            Thread.sleep(10000); // simulate the slow job here . .
            . publishProgress((Long) i);
         }
      } catch (InterruptedException e) {
         Log.e("slow-job interrupted", e.getMessage());
   }  }
      return null;
```

# Wielowątkowość

## Przykład 4: Wykorzystanie klasy AsyncTask – MainActivity

```
      // periodic updates - it is OK to change UI
      @Override
4 →   protected void onProgressUpdate(Long... value)
         { super.onProgressUpdate(value);
         dialog.setMessage(waitMsg + value[0]);
         txtMsg.append("\nworking..." + value[0]);
      }

      // can use UI thread here
5 →   protected void onPostExecute(final Void unused) {

         if (this.dialog.isShowing()) {
            this.dialog.dismiss();
         }

         // cleaning-up, all done
         txtMsg.append("\nEnd Time:"
               + (System.currentTimeMillis() - startingMillis) / 1000);
         txtMsg.append("\nddone!");
      }

   }// AsyncTask

}// MainActivity
```

# Pytania