



# APLIKACJE MOBILNE

## Wykład 08

### GESTY LOGI SYSTEMOWE

dr Artur Bartoszewski

# OBSŁUGA GESTÓW



- ✓ Do obsługi gestów w API Androida służy obiekt GestureDetector

`GestureDetector nazwa = new GestureDetector(kontekst, słuchacz);`

- ✓ GestureDetector posiada interfejs `OnGestureListener` (słuchacz gestów), który implementuje metody odpowiedzialne za rozpoznawanie i obsługę różnego rodzaju gestów.
- ✓ Jedną z metod zaimplementowania słuchacza gestów jest rozszerzenie klasy MainActivity (głównej aktywności) o ten interfejs

```
public class MainActivity extends AppCompatActivity implements GestureDetector.OnGestureListener {
```

```
import ...
```

Select Methods to Implement



android.view.GestureDetector.OnGestureListener

- ☒ onDown(e:MotionEvent):boolean
- ☒ onShowPress(e:MotionEvent):void
- ☒ onSingleTapUp(e:MotionEvent):boolean
- ☒ onScroll(e1:MotionEvent, e2:MotionEvent, distanceX:float, distanceY:float):boolean
- ☒ onLongPress(e:MotionEvent):void
- ☒ onFling(e1:MotionEvent, e2:MotionEvent, velocityX:float, velocityY:float):boolean

android.view.Window.Callback

- ☒ onPointerCaptureChanged(hasCapture:boolean):void

```
MainActivity implements GestureDetector.OnGestureListener {
```

```
    private static final int
```

```
    ;
```

```
    main);
```

## Implementacja metod listenera

W tym przykładzie aktywność **MainActivity** implementuje interfejs **GestureDetector.OnGestureListener**.

Po zaimplementowaniu interfejsu należy uzupełnić aktywność o związane z nim metody

## Implementacja metod listenera

```
@Override
public boolean onDown(MotionEvent e) { return false; }

@Override
public void onShowPress(MotionEvent e) {}

@Override
public boolean onSingleTapUp(MotionEvent e) { return false; }

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY) {...}

@Override
public void onLongPress(MotionEvent e) {}

@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY) {...}
```

Interfejs `OnGestureListener` implementuje wewnątrz klasy `MainActivity` metody obsługi gestów.

## Dodawanie obiektu GestureDetector

```
public class MainActivity extends AppCompatActivity
    implements GestureDetector.OnGestureListener {
    GestureDetector detektorGestow;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        detektorGestow = new GestureDetector( context: this, listener: this);
    }
```

Następnie w metodzie onCreate tworzymy instancję słuchacza gestów – instancję klasy GestureDetector

- Drugim parametrem jego konstruktora jest „this” – czyli prościej mówiąc słuchaczem dla gestów staje się główna aktywność.

## Przechwytywanie zdarzeń i przesyłanie do GestureDetector-a

```
@Override  
public boolean onTouchEvent(MotionEvent event) {  
    detektorGestow.onTouchEvent(event);  
    return super.onTouchEvent(event);  
}
```

Aby przechwycić zdarzenie gestu nadpisujemy metodę: **onTouchEvent(..)**  
Otrzymuje ona w parametrze obiekt **event** typu **MotionEvent** opisujący gest.

Zdarzenie to przesyłamy do zdefiniowanego wcześniej słuchacza gestów za pomocą metody **onTouchEvent()**

np.: `detektorGestow.onTouchEvent(event)`

## Naciśnięcie ekranu - onDown

@Override

```
public boolean onDown(MotionEvent motionEvent) {  
    float X = motionEvent.getX();  
    float Y = motionEvent.getY();  
    float xp = motionEvent.getXPrecision();  
    float silaNacisku = motionEvent.getPressure();  
    opis.setText("onDown: "+ String.valueOf(X)+":"+String.valueOf(Y)+  
                "\n"+String.valueOf(silaNacisku));  
    return false;  
}
```

Metoda reaguje na dotknięcie ekranu (natychmiastowo w momencie dotknięcia)

Obiekt „**motionEvent**” typu **MotionEvent** opisuje gest. Posiada on metody, za pomocą których odczytać możemy parametry gestu np.:

- .getX() .getY() - pozycja dotknięcia
- getPressure() – siła nacisku



## „Tupnięcie” na ekran - onSingleTapUp

```
@Override
public boolean onSingleTapUp(MotionEvent motionEvent) {
    float X = motionEvent.getX();
    float Y = motionEvent.getY();
    opis.setText("onSingleTapUp: \n"+ String.valueOf(X)+":"+String.valueOf(Y));
    return false;
}
```

Metoda reaguje na krótkie stuknięcie w ekran.

Obiekt „**motionEvent**” typu **MotionEvent** opisujący gest posiada metody, za pomocą których odczytać możemy parametry gestu np.: `.getX()` `.getY()`

## Krótkie naciśnięcie ekranu- onShowPress

```
@Override
public void onShowPress(MotionEvent e) {
    float px = e.getX();
    float py = e.getY();
    opis.setText("onShowPress " +
        "\n x="+String.valueOf(px) +
        "\n y="+String.valueOf(py) );
}
```

Obiekt „e” typu **MotionEvent** opisujący gest posiada metody, za pomocą których odczytać możemy parametry gestu np.: `.getX()` `.getY()`

## Długie naciśnięcie ekranu- onLongPress

```
@Override
public void onLongPress(MotionEvent motionEvent) {
    float X = motionEvent.getX();
    float Y = motionEvent.getY();
    opis.setText("onLongPress: \n"
                + String.valueOf(X)+":"+String.valueOf(Y));
}
```

Obiekt „**motionEvent**” typu **MotionEvent** opisujący gest posiada metody, za pomocą których odczytać możemy parametry gestu np.: `.getX()` `.getY()`

## Przeciągnięcie po ekranie

Ges przeciągnięcia po ekranie generuje dwa zdarzenia:

- ✓ **onScrool** – zdarzenie ciągłe – na bieżąco reaguje na ruch
- ✓ **onFiling** – zdarzenie pojedyncze – wywoływane po przeciągnięciu po ekranie
- ✓ Do zdarzeń przekazywane są obiekty ***motionEvent*** (parametry początkowe zdarzenia) i ***motionEven1*** (parametry końcowe zdarzenia). Pozwalają one odczytać pozycję początku i końca przeciągnięcia oraz jego dystans (w rozbiciu na dystans po osi x oraz po osi y (prędkość gestu nie jest istotna)
- ✓ Parametry ***v*** i ***v1*** pozwalają odczytać prędkość gestu – w rozbiciu na prędkość poziomą i pionową

## Przeciągnięcie po ekranie - onScroll

@Override

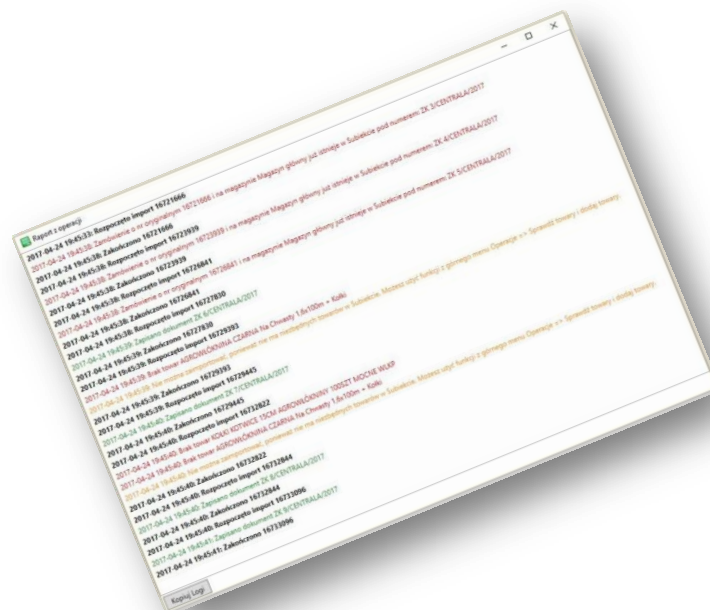
```
public boolean onScroll(MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1) {  
    float X = motionEvent.getX();  
    float Y = motionEvent.getY();  
    float X1 = motionEvent1.getX();  
    float Y1 = motionEvent1.getY();  
    opis.setText("onSingleTapUp: \n"+ String.valueOf(X)+" : "+String.valueOf(Y)+"\n"  
        +String.valueOf(X1)+" ; "+String.valueOf(Y1)+"\n"  
        +String.valueOf(v)+" : "+String.valueOf(v1));  
    return false;  
}
```

## Przeciągnięcie po ekranie - onFling

@Override

```
public boolean onFling(MotionEvent motionEvent, MotionEvent motionEvent1, float v, float v1) {  
    float X = motionEvent.getX();  
    float Y = motionEvent.getY();  
    float X1 = motionEvent1.getX();  
    float Y1 = motionEvent1.getY();  
    opis.setText("onFling: \n"+ String.valueOf(X)+" : "+String.valueOf(Y)+"\n"  
        +String.valueOf(X1)+" ; "+String.valueOf(Y1)+"\n"  
        +String.valueOf(v)+" : "+String.valueOf(v1));  
    return false;  
}
```

## Logi aplikacji



## Rodzaje logów

- ✓ Log.v (..) - VERBOSE
- ✓ Log.d (..) - DEBUG
- ✓ Log.i (..) - INFO
- ✓ Log.w (..) - WARN (warning)
- ✓ Log.e (..) - ERROR

Logi debugowania są kompilowane, ale usuwane w czasie wykonywania. Dzienniki Logi, ostrzeżeń i informacji są zawsze przechowywane.

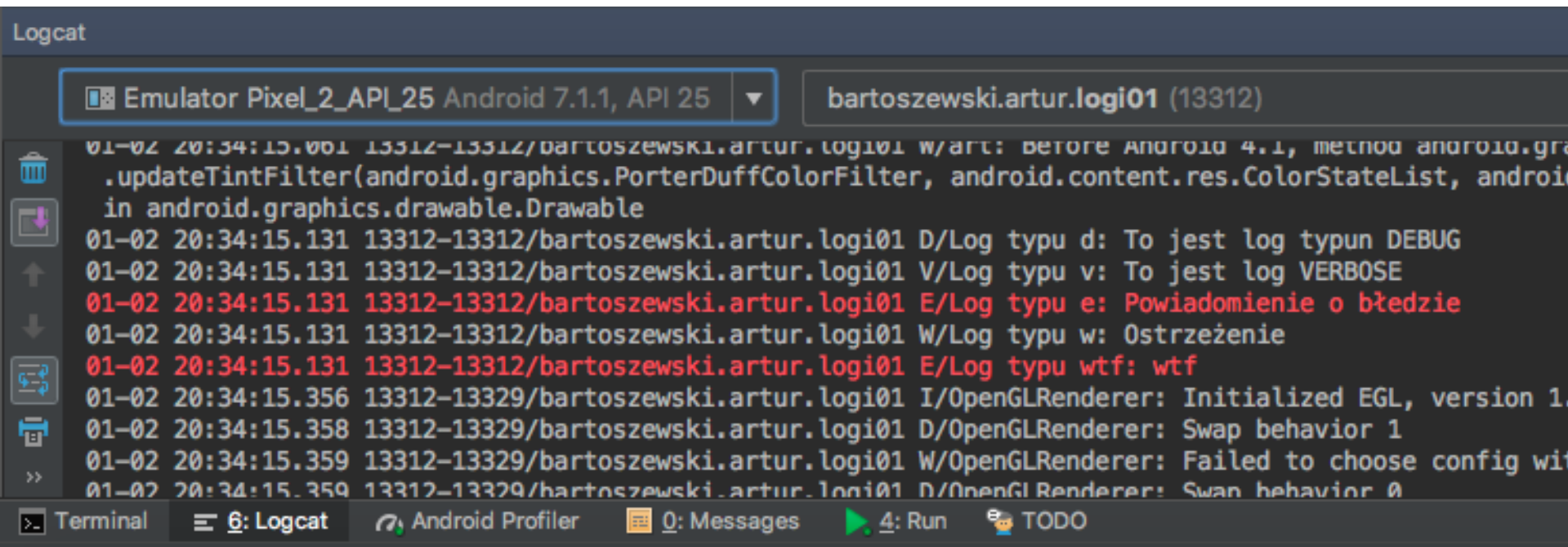


## Tworzenie logów

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Log.d( tag: "Log typu d" , msg: "To jest log typun DEBUG");
    Log.v( tag: "Log typu v" , msg: "To jest log VERBOSE");
    Log.e( tag: "Log typu e" , msg: "Powiadomienie o błędzie");
    Log.w( tag: "Log typu w" , msg: "Ostrzeżenie");
    Log.wtf( tag: "Log typu wtf", msg: "wtf");
}
```

Dla testu logi wygenerowano w metodzie onCreate

## Tworzenie logów



Logcat

Emulator Pixel\_2\_API\_25 Android 7.1.1, API 25    bartoszewski.artur.logi01 (13312)

```
01-02 20:34:15.001 13312-13312/bartoszewski.artur.logi01 W/art: before Android 4.1, method android.graphics.drawable.Drawable.  
.updateTintFilter(android.graphics.PorterDuffColorFilter, android.content.res.ColorStateList, android.graphics.drawable.Drawable)  
in android.graphics.drawable.Drawable  
01-02 20:34:15.131 13312-13312/bartoszewski.artur.logi01 D/Log typu d: To jest log typun DEBUG  
01-02 20:34:15.131 13312-13312/bartoszewski.artur.logi01 V/Log typu v: To jest log VERBOSE  
01-02 20:34:15.131 13312-13312/bartoszewski.artur.logi01 E/Log typu e: Powiadomienie o błędzie  
01-02 20:34:15.131 13312-13312/bartoszewski.artur.logi01 W/Log typu w: Ostrzeżenie  
01-02 20:34:15.131 13312-13312/bartoszewski.artur.logi01 E/Log typu wtf: wtf  
01-02 20:34:15.356 13312-13329/bartoszewski.artur.logi01 I/OpenGLRenderer: Initialized EGL, version 1.1  
01-02 20:34:15.358 13312-13329/bartoszewski.artur.logi01 D/OpenGLRenderer: Swap behavior 1  
01-02 20:34:15.359 13312-13329/bartoszewski.artur.logi01 W/OpenGLRenderer: Failed to choose config with EGL_SWAP_BEHAVIOR_PRESERVED  
01-02 20:34:15.359 13312-13329/bartoszewski.artur.logi01 D/OpenGLRenderer: Swap behavior 0
```

Terminal    6: Logcat    Android Profiler    0: Messages    4: Run    TODO

Logi wygenerowane przez kod z poprzedniego slajdu

# ZADANIE PRAKTYCZNE:

