dr Artur Bartoszewski

PROGRAMOWANIE APLIKACJI MOBILNYCH

Wykład 08:

Cykl życia aktywności

Cykl życia aktywności

Cyklem życia aplikacji nazywamy stany obiektu reprezentującego aplikację przez które przechodzi on od momentu uruchomienia, aż do usunięcia z pamięci operacyjnej.

Cykl życia aplikacji określa jak może się ona zachowywać i do jakich zasobów ma dostęp.

4 podstawowe stany aktywności

1. Aktywny *(ang. active*) – aktywność jest widoczna na pełnym ekranie i jest to jedyna aplikacja jaka jest obecnie otwarta
2. Zapauzowany (*ang. paused*) – użytkownik uruchomił inną aplikację, jednak nie zasłania ona całego ekranu (zapauzowana aplikacja jest nadal częściowo widoczna)
3. Zatrzymany (*ang. stopped*) – użytkownik zminimalizował aplikację lub otworzył inną aplikację, która zajmuje cały ekran.
4. Zniszczony (*ang. destroyed*) – system Android dynamicznie zarządza pamięcią. System może zniszczyć (usunąć z pamięci) aktywność, która jest w stanie zatrzymanym lub zapauzowanym (bardzo rzadko) w celu uzyskania dodatkowej pamięci.

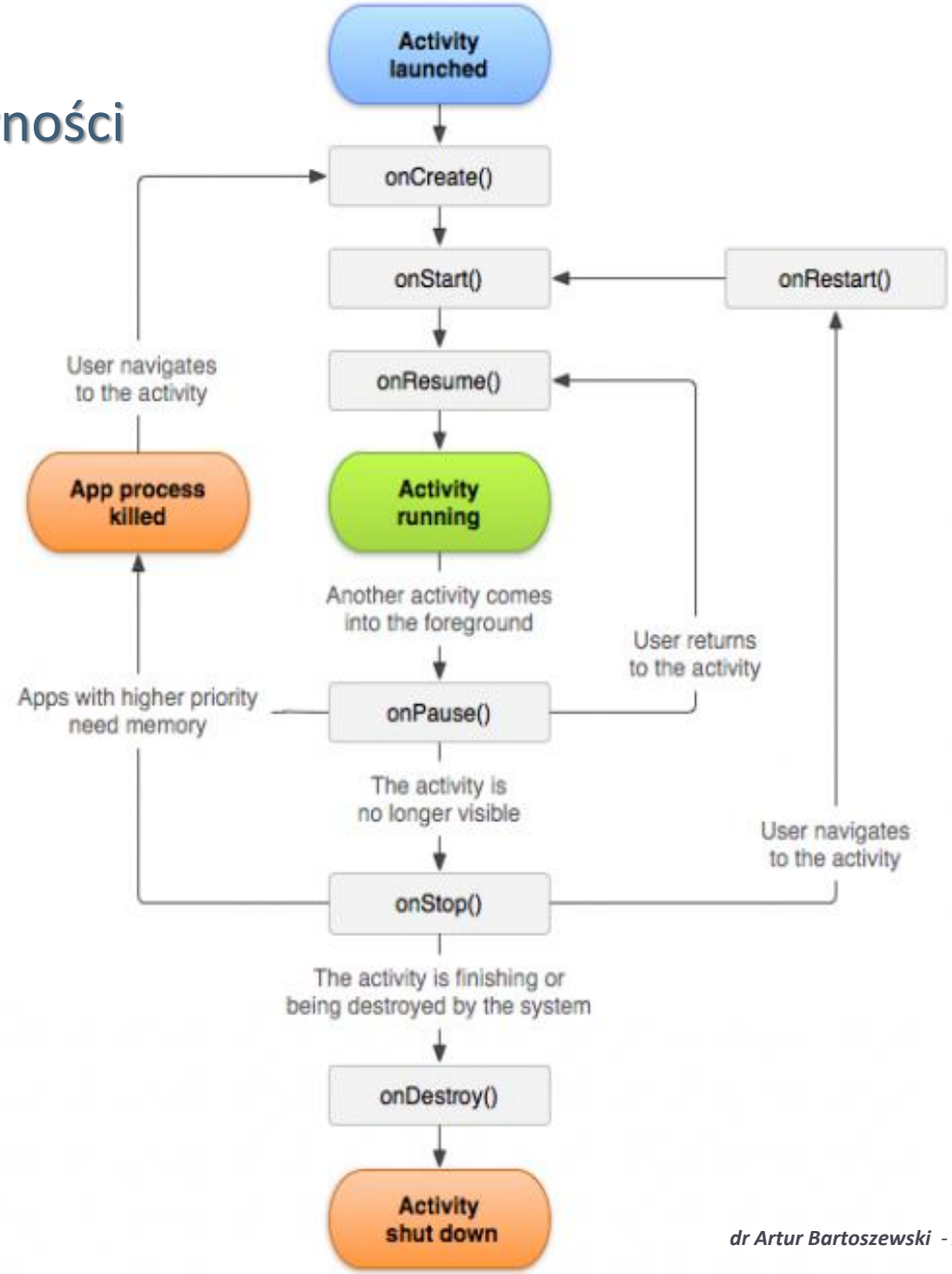
4 stany aktywności

Gdy użytkownik korzysta z aplikacji przechodzi ona przez różne stany cyklu życia. Klasa *Activity* udostępnia podstawowy zestaw siedmiu wywołań zwrotnych:

- `onCreate()`
- `onStart()`
- `onRestart()`
- `onResume()`
- `onPause()`
- `onStop()`
- `onDestroy()`

System wywołuje odpowiednie z nich, gdy proces wchodzi w nowy stan.

Cykl życia aktywności



Nazwa	Opis metody
onCreate()	Metoda wywoływana jest tylko raz w całym cyklu życia aktywności podczas jej uruchamiania. Powinna zawierać inicjalizowanie wszystkich widoków i zmiennych. Przekazywany jest do niej obiekt klasy Bundle zawierający zapisany stan aktywności z poprzedniego uruchomienia (jeżeli istnieje). Po niej wywoływana jest metoda onStart().
onRestart()	Wywoływana po przywróceniu aktywności na ekran - po onStop(). Po niej następuje metoda onStart()
onStart()	Wywoływana przed samym pojawieniem się aktywności na ekranie. Następuje po metodach onCreate() oraz onRestart(). Następną wywoływaną metodą jest onResume()
onResume()	Wywoływana, gdy aktywność jest już na ekranie. Może ją poprzedzać onStart() lub onPause(). W metodzie tej często umieszczane są animacje, a po niej następuje normalna praca aplikacji
onPause()	Wywoływana tuż przed uruchomieniem innej aktywności przez system. Zaleca się zapisywanie istotnych danych już w tej metodzie, gdyż system może zabić aktywność bez wywoływania metod onStop() i onDestroy(). Po tej metodzie wywołana zostanie onResume() lub onStop()
onStop()	Następuje po onPause(), gdy aplikacja nie jest już widoczna. Należy pamiętać, że metoda ta może w ogóle nie być wywołana gdy aktywność zostanie usunięta z pamięci już po onPause(). Po niej następuje onRestart() lub onDestroy()
onDestroy()	Wywoływana przed usunięciem aplikacji z pamięci. Nie można mieć pewności, że metoda ta zostanie w ogóle wywołana

Źródło: http://programowanie.opole.pl

Cykl życia aktywności

Aktywność aplikacji (od stworzenia do zamknięcia)

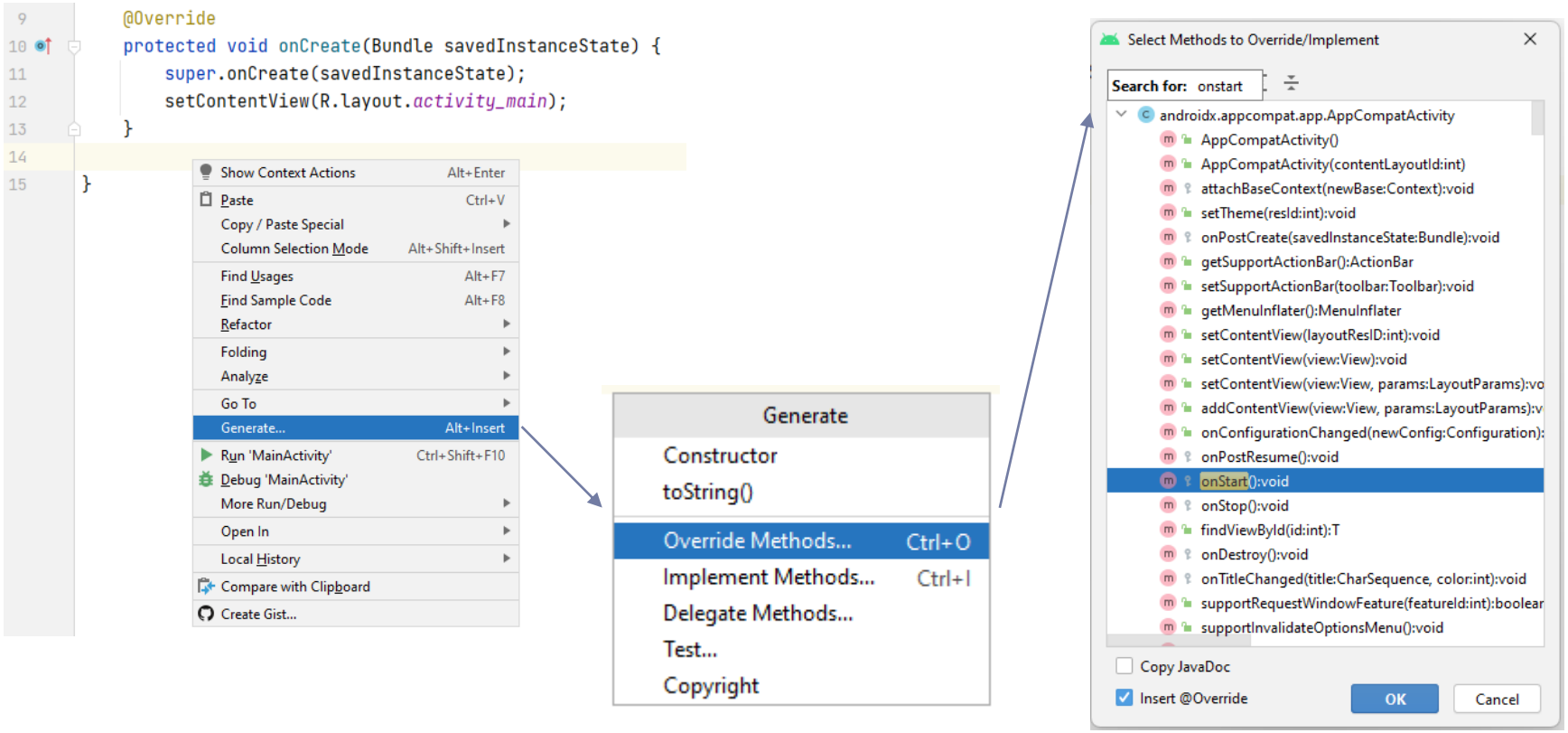
1. Aplikacja zostaje uruchomiona, czyli zostaje utworzona instancja obiektu Aktywność (system tworzy obiekt aktywności poprzez uruchomienie jego konstruktora).
2. Po uruchomieniu obiektu aktywności zostaje wywołana metoda onCreate(), w której ciele należy umieścić kod który ma się wykonać wraz ze startem aplikacji.
3. Następnym cyklem jest działanie aktywności, w trakcie kiedy jest włączona na ekranie i użytkownik może prowadzić z nią interakcję (jest to główny stan aplikacji).
4. Metoda onDestroy() w aktywności jest wywoływana tuż przed zamknięciem aplikacji (pozwala ona na zwolnienie używanych przez nią zasobów).
5. Po wykonaniu metody onDestroy() aktywność przestaje istnieć, a aplikacja zostaje zamknięta.

Cykl życia aktywności

Metody onStop(), onPause() i onResume()

1. Metoda onStop() jest uruchamiana, podczas gdy aplikacja jest zamykana.
2. Metoda onPause() istnieją trzy momenty jej wywołania:
   - jest wywoływana przed metodą onStop() w przypadku zamykania aplikacji.
   - jest wywoływana, gdy użytkownik przejdzie do innej aktywności
   - jest wywołania w momencie, w którym obracamy ekran smartfona.
3. Metoda onResume() jest wywoływana tak jak metoda onPause() w momencie, gdy obracamy ekran smartfona, lub przy powrocie do działania po wcześniejszym wywołaniu metody onPause()

Cykl życia aktywności



Klasa Activity po której dziedziczy aplikacja posiada komplet obsługi metod zdarzeń cyklu życia dlatego należy nadpisać te z nich których które chcemy samodzielnie obsłużyć

Ponieważ metod na liście jest bardzo dużo najlepiej skorzystać z wyszukiwania

Cykl życia aktywności

```
@Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //Aktywność jest tworzona
    }
    @Override
    protected void onStart() {
        super.onStart();
        //Aktywność jest przed ukazaniem się na ekranie
    }
    @Override
    protected void onResume() {
        super.onResume();
        //Aktywność jest już na ekranie
    }
    @Override
    protected void onPause() {
        super.onPause();
        //Aktywność jest przed przysłonięciem przez inną Aktywność
    }
    @Override
    protected void onStop() {
        super.onStop();
        //Aktywność nie jest już widoczna
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        //Aktywność jest przed usunięciem z pamięci
    }
```

Cykl życia aktywności

W systemie Android istnieje wiele scenariuszy, którym odpowiadają określone sekwencje przejść pomiędzy stanami aplikacji.

Przykładowo, gdy nastąpi **zmiana orientacji ekranu z pionowej na poziomą**, aktywność jest niszczona i ponownie odtwarzana. Zamykana jest aktywność związana z pionowym układem ekranu i wywoływane są dla niej zdarzenia:

```
onPause() -> onStop() -> onDestroy()
```

Następnie tworzona jest nowa aktywność (związana z poziomym układem ekranu) dla której wywołane są zdarzenia:

```
onCreate() -> onStart() -> onResume()
```

Efektem tej sekwencji zdarzeń może być reset stanu wszystkich komponentów widocznych na ekranie i utrata danych zapisanych w komponentach częściowo już wypełnionego formularza.

SharedPreferences

SharedPreferences umożliwiają przechowywanie i zarządzanie prostymi danymi.

Dane przechowywane są na urządzeniu w plików xml niezależnie od cyklu życia aplikacji.

Istnieją dopóki nie zostaną usunięte przez kod programu lub wyczyszczone ręcznie przez użytkownika z danych aplikacji.

Aplikacja może posiadać wiele instancji SharedPreferences, które najczęściej są prywatne lecz mogą być także publiczne dla innych aplikacji.

SharedPreferences

W SharedPreferences możemy przechowywać tylko obiekty typu boolean, float, int, long, String.

Metoda zapisywania danych	Metoda odczytywania danych	Typ danych
`putBoolean()`	`getBoolean()`	`boolean`
`putFloat()`	`getFloat()`	`float`
`putInt()`	`getInt()`	`int`
`putLong()`	`getLong()`	`long`
`putString()`	`getString()`	`String`
`putStringSet()`	`getStringSet()`	`Set<String>`

SharedPreferences

Zapis danych

Rozpoczynamy od uzyskania referencji do obiektu typu SharedPreferences:

- odnajdujemy obiekt danej aktywności (jeżeli kod piszemy w jednej z metod MainActivity podajemy this),
- wywołujemy na nim metodę getPreferences() z argumentem Context.MODE_PRIVATE (dla referencji prywatnych programu),

1. Tworzymy obiekt typu SharedPreferences.Editor wywołując metodę edit() na obiekcie preferencji,
2. Na nowo utworzonym obiekcie wywołujemy metodę putString(). Jej parametrami są dane do zapisania w formie „klucz->wartość”
3. Zatwierdzamy zmiany metodą commit().

```
SharedPreferences sharedPreferences =
        this.getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor edytor = sharedPreferences.edit();
edytor.putString("klucz","dane do zapamiętania");
edytor.commit();
```

SharedPreferences

Odczyt danych

1. Rozpoczynamy od uzyskania referencji do obiektu typu SharedPreferences:
   - odnajdujemy obiekt danej aktywności (jeżeli kod piszemy w jednej z metod MainActivity podajemy this),
   - wywołujemy na nim metodę getPreferences() z argumentem Context.MODE_PRIVATE (dla referencji prywatnych programu),
2. Wywołujemy na nim odpowiednią metodę – w tym przykładzie getString(),
   - jako pierwszy argument podajemy klucz (wpisany przy ich zapisywaniu),
   - Jako drugi domyślną wartość, która będzie wczytana kiedy plik preferencji nie będzie zawierał takiego klucza.

```
SharedPreferences sharedPreferences = this.getPreferences(Context.MODE_PRIVATE);

String tekst = sharedPreferences.getString("klucz", "");
```

Usuwanie preferencji

1. Do usuwania preferencji służy metoda clear(), którą wywołujemy na obiekcie typu SharedPreferences.editor.

```
SharedPreferences sharedPreferences = this.getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor edytor = sharedPreferences.edit();
edytor.clear();
edytor.commit();
```

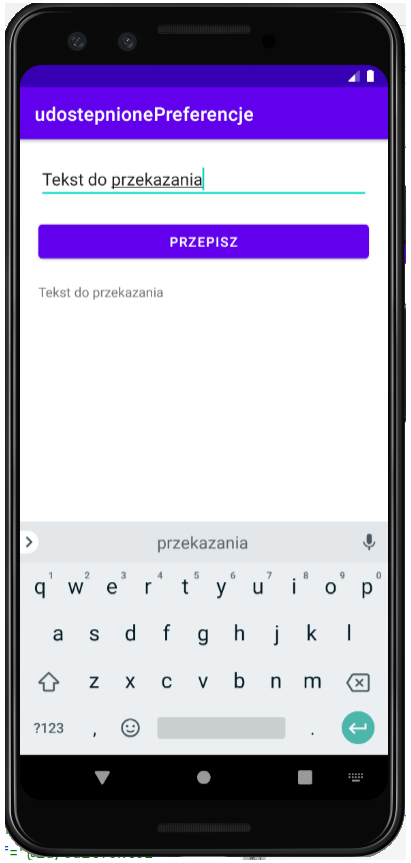
SharedPreferences

Przykład:

Przekazywanie danych pomiędzy stanami życia aplikacji.

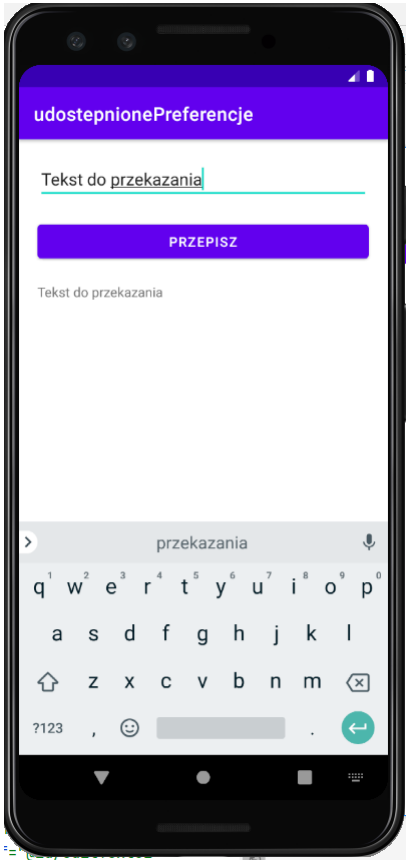
Pozornie program jest banalny.
Po kliknięciu na przycisk tekst z okna tekstowego powyżej powinien być przepisany do kontrolki TekstVew poniżej przycisku.

Jednak jeżeli spróbujemy obrócić telefon napotkamy problem: przepisany do pola tekstowego tekst zniknie



Przykład

Interface programu




```
<EditText
    android:id="@+id/editText01"
    android:hint="Tu wpisz tekst"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginTop="20dp"
    android:layout_marginLeft="20dp"
    android:layout_marginRight="20dp"/>
<Button
    android:id="@+id/button01"
    android:text="Przepisz"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="@id/editText01"
    app:layout_constraintRight_toRightOf="@id/editText01"
    app:layout_constraintTop_toBottomOf="@id/editText01"
    android:layout_marginTop="20dp"
    />
<TextView
    android:id="@+id/textView01"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="@id/editText01"
    app:layout_constraintRight_toRightOf="@id/editText01"
    app:layout_constraintTop_toBottomOf="@id/button01"
    android:layout_marginTop="20dp"/>
```

Przykład

```
EditText editText01;
TextView textView01;
Button button01;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    editText01 = findViewById(R.id.editText01);
    textView01 = findViewById(R.id.textView01);
    button01 = findViewById(R.id.button01);
    button01.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            textView01.setText(editText01.getText().toString());
        }
    });
}
```

Na tym etapie aplikacja działa, lecz występuje opisany wyżej błąd.
Wynika on z tego, że w trakcie obracania ekranu aktywność jest usuwana i tworzona od nowa.

Przykład

Rozwiązaniem jest zapamiętanie danych w metodzie z obsługi zdarzenia onPause() i ich ponowne wczytanie w metodzie onResume()

```
@Override
protected void onPause() {
    super.onPause();
    SharedPreferences sharedPreferences = this.getPreferences(Context.MODE_PRIVATE);
    SharedPreferences.Editor edytor = sharedPreferences.edit();
    edytor.putString( s: "dane",textView01.getText().toString());
    edytor.commit();
}
```

```
@Override
protected void onResume() {
    super.onResume();
    SharedPreferences sharedPreferences = this.getPreferences(Context.MODE_PRIVATE);
    String tekst = sharedPreferences.getString( s: "dane",  s1: "");
    textView01.setText(tekst);
}
```

Na marginesie

W następnym przykładzie wykorzystywać będziemy datę i czas systemowy.

Istnieje kilka klas służących do obsługi daty i czasu.

W starszych wersjach Javy:

```
Date data = new Date();
SimpleDateFormat prostyFormatDaty =
            new SimpleDateFormat("YYYY-mm-dd HH:mm:ss");
String wynik = prostyFormatDaty.format(data);
```

Pobieramy czas systemowy w formacie Time Stump (czas Unixowy), a następnie konwertujemy go na czytelny dla użytkownika format przy użyciu jednego z dostępnych wzorców.

Na marginesie

Od 8 wersji Javy mamy do dyspozycji klasy LocalDate i LocalTime

```
LocalTime localTime = LocalTime.now();
String czas = localTime.toString();
int godzina = localTime.getHour();
int minuta = localTime.getMinute();
int sekunda = localTime.getSecond();
int nanosekunda = localTime.getNano();
```

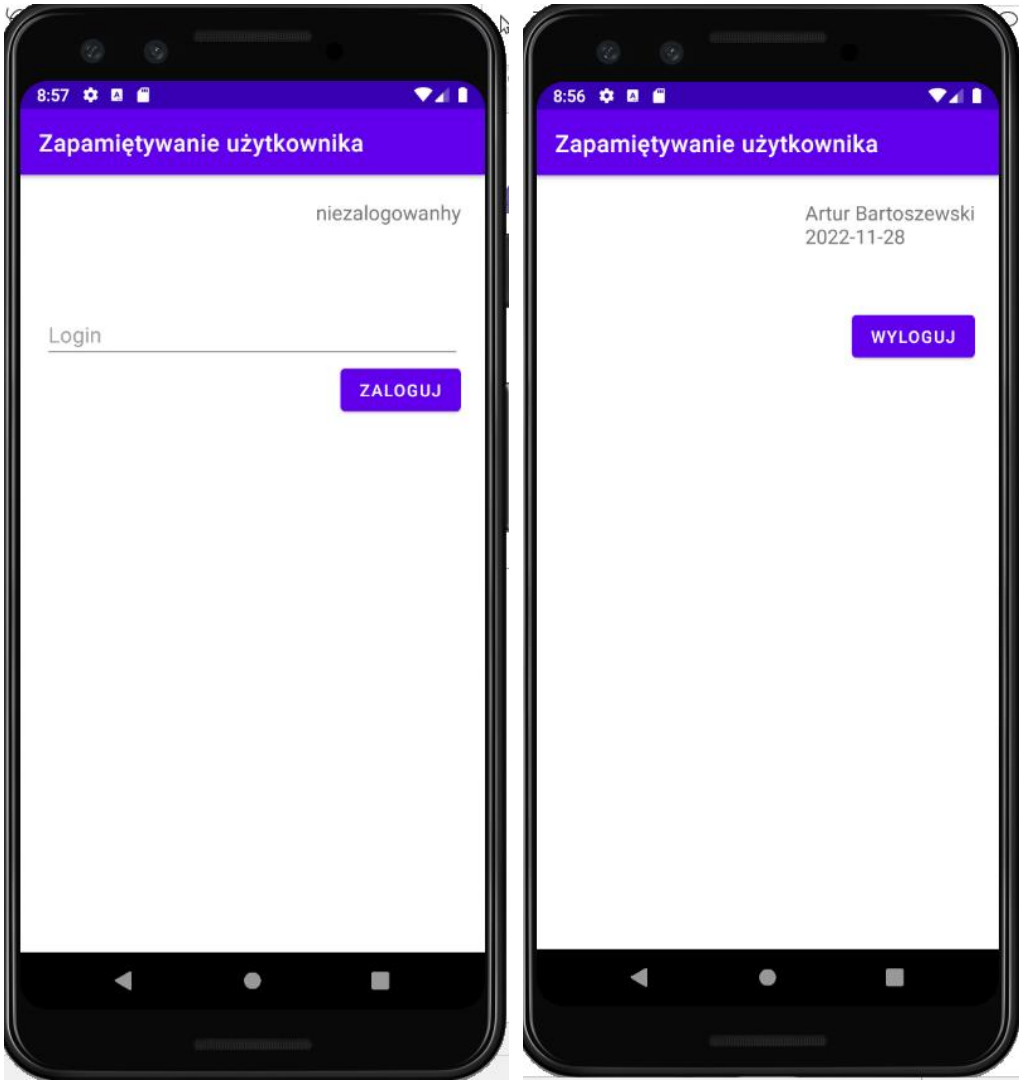
Tworzymy obiekt klasy LocalTime i pobieramy aktualny czas systemowy za pomocą metody .now()

Podobnie dział klasa LocalDate.

```
LocalDate localDate = LocalDate.now();
String dataTekstowo = localDate.toString();
int rok = localDate.getYear();
int dzienWRoku = localDate.getDayOfYear();
```

Przykład:

- Program powinien umożliwić zalogowanie się. Zapamiętać zalogowanego użytkownika nawet po wyłączeniu aplikacji.
- Program powinien pamiętać datę jego zalogowania.
- Umożliwić ręczne wylogowanie się



Przykład:

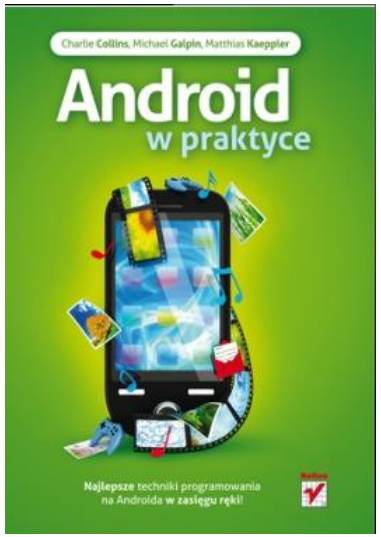
```
17  public class MainActivity extends AppCompatActivity {
18
19      EditText editText01;
20      TextView textView01;
21      Button bt_zaloguj, bt_wyloguj;
22      SharedPreferences sharedPreferences;
23      SharedPreferences.Editor edytorPreferencji;
24      @Override
25      protected void onCreate(Bundle savedInstanceState) {
26          super.onCreate(savedInstanceState);
27          setContentView(R.layout.activity_main);
28          editText01 = findViewById(R.id.ediText01);
29          textView01 = findViewById(R.id.textView01);
30          bt_wyloguj = findViewById(R.id.bt_wyloguj);
31          bt_zaloguj = findViewById(R.id.bt_zaloguj);
32          sprawdzZalogowanie();
33          dodajSluchacze();
34      }
```

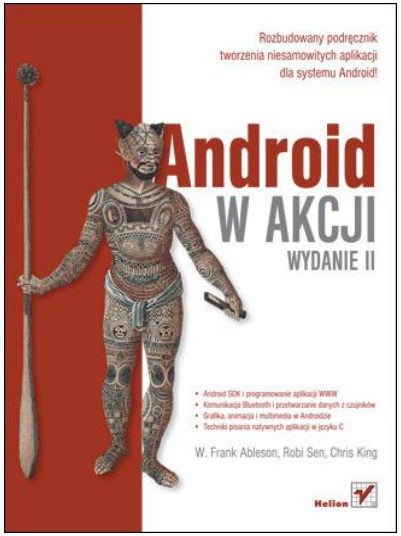
Przykład:

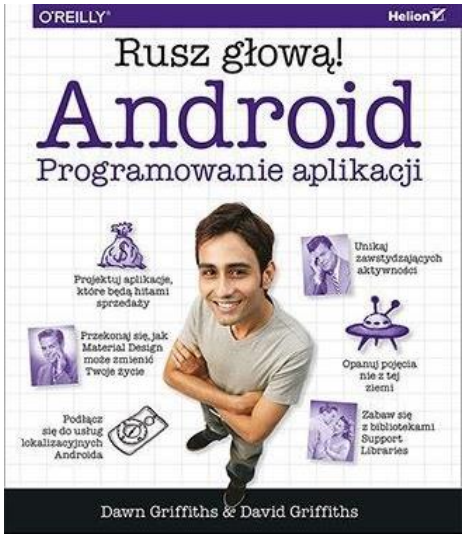
```
36      private void sprawdzZalogowanie() {
37          sharedPreferences = this.getPreferences(Context.MODE_PRIVATE);
38          edytorPreferencji = sharedPreferences.edit();
39          String login = sharedPreferences.getString( s: "login", s1: "");
40          String data = sharedPreferences.getString( s: "data", s1: "");
41          if(login.length()>0)
42          {
43              login += "\n";
44              login += data;
45              textView01.setText(login);
46              editText01.setVisibility(View.INVISIBLE);
47              bt_zaloguj.setVisibility(View.INVISIBLE);
48              bt_wyloguj.setVisibility(View.VISIBLE);
49          }
50          else
51          {
52              textView01.setText("niezalogowanhy");
53              editText01.setVisibility(View.VISIBLE);
54              bt_zaloguj.setVisibility(View.VISIBLE);
55              bt_wyloguj.setVisibility(View.INVISIBLE);
56          }
57      }
```

```
59      private void dodajSluchacze() {
60          View.OnClickListener sluchacz = new View.OnClickListener() {
61              @RequiresApi(api = Build.VERSION_CODES.O)
62              @Override
63              public void onClick(View view) {
64                  int Id = view.getId();
65                  String login = editText01.getText().toString();
66                  if(login.length()==0) {login="anonim";}
67
68                  if(Id==R.id.bt_zaloguj)
69                  {
70                      edytorPreferencji.putString( s: "login",login);
71                      LocalDate localDate = LocalDate.now();
72                      edytorPreferencji.putString( s: "data",localDate.toString());
73                  }
74                  else if(Id==R.id.bt_wyloguj)
75                  {
76                      edytorPreferencji.clear();
77                  }
78                  edytorPreferencji.commit();
79                  sprawdzZalogowanie();
80              }
81          };
82          bt_zaloguj.setOnClickListener(sluchacz);
83          bt_wyloguj.setOnClickListener(sluchacz);
84      }
```

Literatura


Charlie Collins, Michael Galpin, Matthias Kaeppler
Android
w praktyce
Najlepsze techniki programowania na Androida w zasięgu ręki!


Rozbudowany podręcznik tworzenia niesamowitych aplikacji dla systemu Android!
Android
W AKCJI
WYDANIE II
W. Frank Ableson, Robi Sen, Chris King
Helion


O'REILLY
Helion
Rusz głową!
Android
Programowanie aplikacji
Projektuj aplikacje, które będą hitami sprzedaży
Przekonaj się, jak Material Design może zmienić Twoje życie
Podłącz się do usług lokalizacyjnych Androida
Unikaj zawstydzających aktywności
Opanuj pojęcia nie z tej ziemi
Zabaw się z bibliotekami Support Libraries
Dawn Griffiths & David Griffiths