Cykl życia aktywności

mgr inż. Stanisław Lota

Wprowadzenie

Aktywność to najważniejsza rzecz każdej aplikacji. Możemy przyjąć, że **jest to ekran, który zajmuje powierzchnię całego wyświetlacza smartfonu.**

Aktywność przedstawia pojedynczy ekran aplikacji, z którym użytkownik może w danym momencie przeprowadzać interakcje.

W aktywności możemy umieszczać mnóstwo innych elementów takich, jak przyciski, obrazki i tym podobne.

Wprowadzenie

Każda aplikacja może mieć kilka Aktywności (każda z własnym cyklem życia). Każda aktywność posiada szereg wewnętrznych stanów (tworzenie, uruchamianie, zawieszenie, wznawianie, niszczenie).

W zależności od działań podejmowanych przez użytkownika, aktywność zmienia stan i automatycznie uruchamiana jest przez system odpowiednia metoda cyklu życia aktywności.

Cykl życia Androida to jedno z najczęściej zadawanych pytań rekrutacyjnych!

Wprowadzenie

Aktywności mogą być wywoływane bezpośrednio przez użytkownika (za pomocą lauchera), ale również jedna aktywność może uruchomić drugą.

Możliwe jest nawet wywoływanie aktywności należących do innych aplikacji (np. aplikacji do wysyłania e-maili czy obsługi aparatu fotograficznego).

Ze względu na to, że ani programista aplikacji, ani jej użytkownik nie mają stuprocentowej kontroli nad cyklem życia aktywności, ważne jest poprawne zapewnienie obsługi wszystkich możliwych sytuacji.

Wprowadzenie

Aplikacja na Androida różni się od aplikacji na komputery stacjonarne. Ta różnica polega na tym, że aplikacja na Androida nie zawsze zaczyna się w tym samym miejscu. Ponadto nie ma określonego początku dla aplikacji na Androida.

Wprowadzenie

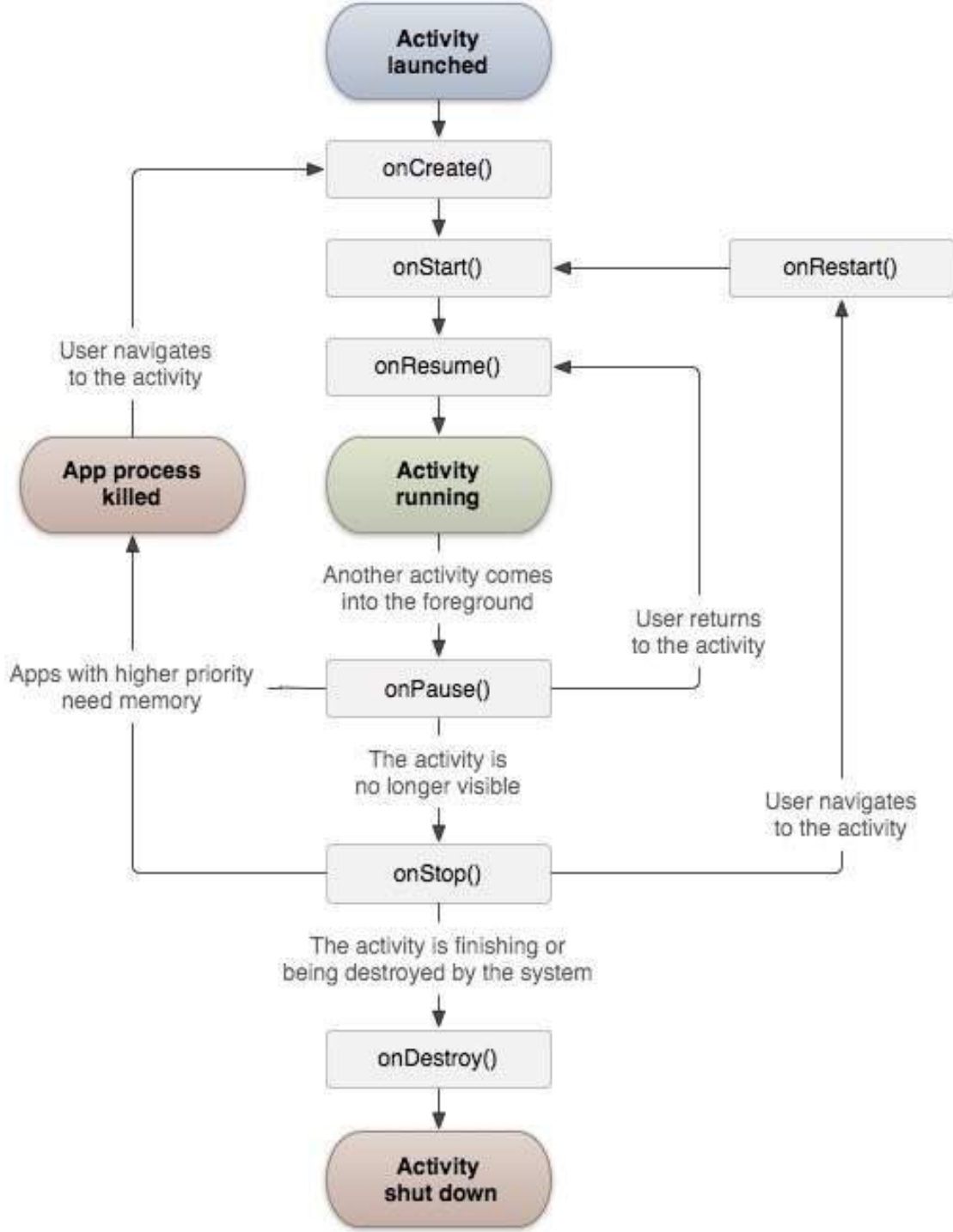
Wystąpienia działania w aplikacji zmieniają się w różnych stanach w ich cyklu życia. Ponadto możemy stwierdzić, że system Android tworzy, zatrzymuje, wznawia działanie lub niszczy cały proces za pomocą szeregu **metod wywołania zwrotnego** udostępnianych przez klasę activity. Gdy użytkownik nawiguje, wychodzi z aplikacji i wraca do niej.

Akcje związane z cyklem życia Activity

Entire Lifetime – całe życie, pomiędzy wywołaniem metody onCreate() a onDestroy().

Visible Lifetime – pomiędzy onStart() i onStop(), aktywność jest widoczna dla użytkownika, ale nie jest z nim w interakcji.

Foreground Lifetime – pomiędzy onResume() i onPause() , aktywność jest w interakcji z użytkownikiem.


Activity launched
onCreate()
onStart()
onRestart()
onResume()
User navigates to the activity
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Cykl życia aktywności

Korzystając z metod wywołania zwrotnego cyklu życia, możemy określić, jak zachowuje się aktywność, gdy użytkownik opuszcza i ponownie wchodzi do działania.

Na przykład podczas tworzenia aplikacji do odtwarzania strumieniowego wideo należy wstrzymać wideo i zakończyć połączenie sieciowe, gdy użytkownik przełączy się na inną aplikację.

Cykl życia aktywności

Dobra implementacja metod wywołania zwrotnego cyklu życia pozwala na wykonanie określonej pracy, która jest odpowiednia dla danej zmiany stanu oraz wykonywanie odpowiedniej pracy we właściwym czasie i odpowiedniej obsługi przejść. Dzięki niej sprawiają, że aplikacja jest bardziej niezawodna i wydajna.

Cykl życia aktywności

Aplikacja będzie również unikać:

1. Awarii jeśli użytkownik odbierze połączenie telefoniczne lub przełączy się na inną aplikację podczas korzystania z niej.
2. Awarii lub utraty postępu użytkownika, gdy ekran obraca się między orientacją poziomą i pionową.
3. Zużywania cennych zasobów systemowych, gdy użytkownik nie korzysta z aplikacji aktywnie.
4. Utraty postępów użytkownika, jeśli opuści on aplikację i wróci do niej w późniejszym czasie.

Cykl życia aktywności

Łącznie istnieje siedem wywołań zwrotnych onCreate(), onStart(), onResume (), onPause (), onStop (), onDestroy () i onRestart ().

Pierwszą fazą życia Activity jest jej **utworzenie.** Kolejnym etapem jest **faza działania**. Aktywność działa, kiedy jest wyświetlona na ekranie i znajduje się na pierwszym planie (użytkownik może z nią prowadzić interakcje). Ostatnią fazą życia Activity jest śmierć czyli **zostanie usunięta (lub zniszczona)**.

onCreate()

Ta metoda jest uruchamiana, gdy aplikacja tworzy działanie i musimy użyć tej metody. Konieczne będzie również dodanie podstawowej logiki uruchamiania aplikacji, która powinna mieć miejsce tylko raz w całym cyklu życia działania.

Aktywność nie utrzymuje się w stanie on_Create, więc po wykonaniu metody onCreate system odpala metody onStart () i onResume ().

onStart()

Metoda onStart wywoływana jest, gdy działanie przejdzie w stan początkowy. Wywołanie metody sprawia, że działanie jest widoczne dla użytkownika, ponieważ aplikacja przygotowuje się do wejścia na pierwszy plan i interaktywności.

Na przykład ta metoda polega na tym, że aplikacja inicjuje kod, który obsługuje interfejs użytkownika. Ta metoda jest wykonywana szybko i po przejściu aktywności do stanu wznowienia.

onResume()

Ta metoda jest wywoływana, gdy działanie przechodzi w stan wznowienia, w którym aplikacja jest gotowa do interakcji z użytkownikiem na pierwszym planie.

Aplikacja pozostaje w tym stanie, dopóki coś nie odwróci jej uwagi. Takim zdarzeniem może być na przykład odebranie telefonu, przejście użytkownika do innej czynności lub wyłączenie ekranu urządzenia.

onResume()

W przypadku wystąpienia zdarzenia przerywającego działanie przechodzi w stan *wstrzymania* , a system wywołuje wywołanie zwrotne onPause()

onResume()

Jeśli działanie powróci do stanu Resumed ze stanu Paused, system ponownie wywoła onResume() metodę method.

Z tego powodu należy zaimplementować, onResume() aby zainicjować komponenty, które zostały zwolnione w trakcie onPause(), i wykonać wszelkie inne inicjalizacje, które muszą nastąpić za każdym razem, gdy działanie przechodzi w stan wznowienia.

onPause()

Gdy użytkownik przejdzie do trybu wielu okien (kliknie przycisk zadania), ta metoda jest wywoływana przez system. Ponadto wskazuje, że użytkownik opuszcza aktywność.

Co więcej, możemy użyć tej metody do wstrzymania lub dostosowania operacji, które nie powinny być kontynuowane, gdy działanie jest w stanie wstrzymania i które wkrótce zostaną wznowione.

onPause()

Istnieje kilka powodów, dla których aktywność może wejść w ten stan. Na przykład:

- Niektóre zdarzenia przerywają wykonywanie aplikacji. To jest najczęstszy przypadek.
- W systemie Android 7.0 (poziom interfejsu API 24) lub nowszym wiele aplikacji działa w trybie wielu okien. Ponieważ tylko jedna z aplikacji (okien) jest aktywna w dowolnym momencie, system wstrzymuje wszystkie pozostałe aplikacje.

onPause()

- Otworzy się nowe, półprzezroczyste działanie (na przykład okno dialogowe). Dopóki czynność jest nadal częściowo widoczna, ale nie jest wyostrzona, pozostaje wstrzymana.
- Zakończenie działania funkcji onPause () nie oznacza, że operacja opuszcza warunek wstrzymania. Zamiast tego operacja utrzymuje się w tym stanie, dopóki działanie nie powróci lub nie stanie się całkowicie niedostępne dla użytkownika.

onPause()

Jeśli aktywność zostanie wznowiona, system ponownie wywoła wywołanie zwrotne onResume (). Jeśli działanie powróci ze stanu wstrzymania do stanu wznowienia, system zachowa instancję działania w pamięci, przywołując tę instancję, gdy system wywoła metodę onResume ().

onPause()

W takim przypadku nie musimy ponownie inicjować komponentów, które zostały wygenerowane podczas którejkolwiek z metod wywołania zwrotnego prowadzącego do stanu Revived.

Jeśli aktywność stanie się całkowicie niewidoczna, system wywoła metodę onStop ().

onStop ()

Gdy aktywność nie jest już dostępna dla użytkownika, osiągany jest stan Stop i urządzenie wywołuje wywołanie zwrotne onStop ().

Nastąpi to na przykład wtedy, gdy nowo zwolniona aktywność wypełni cały ekran. System może nawet wywołać metodę onStop (), gdy operacja zakończy się i ma zostać zakończona.

onStop()

Możemy także użyć onStop () do wykonywania operacji zamykania, które znacznie obciążają procesor. Na przykład, jeśli nie możemy znaleźć bardziej aktualnego czasu na zapisanie informacji w bazie danych, możemy to zrobić, gdy wywołana zostanie funkcja onStop ().

Ze stanu Zatrzymano aktywność wznawia komunikację z użytkownikiem lub kończy działanie i znika.

Jeśli operacja powróci, maszyna wywoła onRestart(). Jeśli operacja się zakończy, maszyna wywoła metodę onDestroy ().

onDestroy ()

Wywołanie zwrotne metody onDestroy () jest wywoływane, zanim aktywność zostanie zniszczona.

System wywołuje to wywołanie zwrotne, ponieważ: czynność dobiega końca (wskutek całkowitego zwolnienia przez użytkownika czynności lub finish() wezwania do wykonania czynności) lub system tymczasowo niszczy aktywność z powodu zmiany konfiguracji (np. obrót urządzenia lub tryb wielu okien).

onDestroy ()

Jeśli działanie dobiega końca, onDestroy () jest ostatnim wywołaniem zwrotnym cyklu życia, które otrzymuje działanie. Jeśli onDestroy () zostanie wywołany w wyniku zmiany konfiguracji, system natychmiast utworzy nową instancję działania, a następnie wywoła onCreate() tę nową instancję w nowej konfiguracji.

Metoda onDestroy() powinna zwolnić wszystkie zasoby, które nie zostały jeszcze wydane przez wcześniejszych wywołań zwrotnych, takich jak onStop().

MainActivity.kt – Activity lifecycle

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d(TAG, "onCreate: " + "The onCreate method callback")
    }

    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart: " + "The onStart method callback")
    }
```

MainActivity.kt – Activity lifecycle

```
override fun onResume() {
    super.onResume()
    Log.d(TAG, "onResume: " + "The onResume method callback")
}

override fun onPause() {
    super.onPause()
    Log.d(TAG, "onPause: " + "The onPause method callback")
}
```

MainActivity.kt – Activity lifecycle

```
override fun onStop() {
    super.onStop()
    Log.d(TAG, "onStop: " + "The onStop method callback")
}

override fun onDestroy() {
    super.onDestroy()
    Log.d(TAG, "onDestroy: " + "The onDestroy method callback")
}

companion object {
    private const val TAG = "MainActivityCallBacks"
}
```