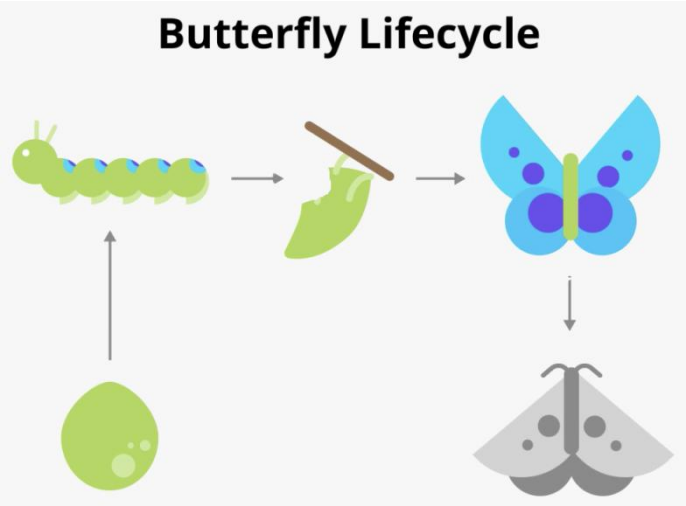
2. Cykl życia aplikacji cz.1

W tym ćwiczeniu będziemy pracować z aplikacją startową o nazwie DessertClicker. Za każdym razem, gdy użytkownik kliknie deser na ekranie „kupuje” go w aplikacji. Aplikacja aktualizuje na ekranie liczbę zakupionych deserów i całkowitą kwotę wydaną przez użytkownika.

Ta aplikacja zawiera kilka błędów związanych z cyklem życia Androida: Na przykład w niektórych okolicznościach aplikacja resetuje wartości dla ilości deserów do 0, oraz korzysta nadal z zasobów systemowych, nawet gdy jest w tle. Zrozumienie cyklu życia Androida pomoże ci zrozumieć, dlaczego występują te problemy i jak je naprawić.

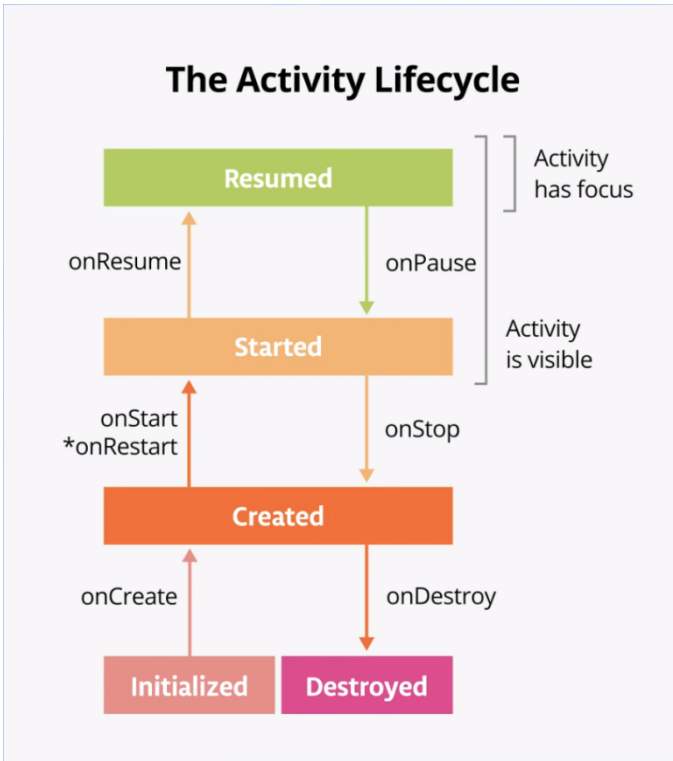
3. Zadanie: poznaj metody cyklu życia i dodaj ich podstawowe monitorowanie

Każde activity i każdy fragment ma tak zwany cykl życia *lifecycle*. Jest to nawiązanie do cykli życia zwierząt, takich jak cykl życia motyla - różne stany motyla pokazują jego wzrost od narodzin do w pełni ukształtowanej dorosłości aż do śmierci.

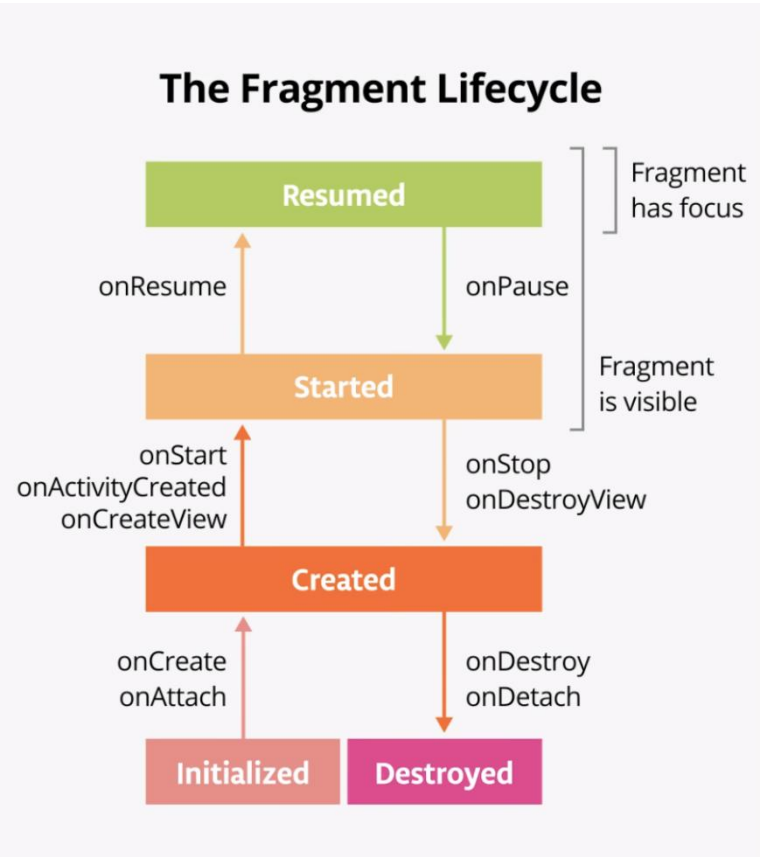


Podobnie, cykl życia activity składa się z różnych stanów, przez które activity może przejść, od momentu pierwszego zainicjowania activity do momentu jego ostatecznego zniszczenia i odzyskania pamięci przez system. Gdy użytkownik uruchamia aplikację, nawiguje między activity, nawiguje wewnątrz i na zewnątrz aplikacji oraz opuszcza aplikację, aktywność zmienia stan. Poniższy schemat pokazuje wszystkie stany cyklu życia aktywności. Jak wskazują ich nazwy, stany te reprezentują status activity.

Często chcesz zmienić pewne zachowanie lub uruchomić kod, gdy zmienia się stan cyklu życia activity. Dlatego sama klasa Activity i wszelkie podklasy Activity, takie jak AppCompatActivity, implementują zestaw metod wywołania zwrotnego cyklu życia. System Android wywołuje te wywołania zwrotne, gdy aktywność przechodzi z jednego stanu do drugiego, i można zastąpić te metody we własnych activity, aby wykonać zadania w odpowiedzi na zmiany stanu cyklu życia. Poniższy diagram pokazuje stany cyklu życia wraz z dostępnymi możliwymi do zastąpienia wywołaniami zwrotnymi.



Fragment ma również cykl życia. Cykl życia fragmentu jest podobny do cyklu życia activity, więc wiele z tego, czego się nauczysz, dotyczy obu. W tym kodzie skupiasz się na cyklu życia aktywności, ponieważ jest to podstawowa część Androida i najłatwiejsza do zaobserwowania w prostej aplikacji. Oto odpowiedni schemat cyklu życia fragmentu:



Ważne jest, aby wiedzieć, kiedy te wywołania zwrotne są wywoływane i co robić w każdej metodzie wywołania zwrotnego. Ale oba te diagramy są złożone i mogą być mylące. W tym ćwiczeniu zamiast po prostu czytać, co oznacza każdy stan oraz funkcja i wywołanie zwrotne, wykonasz trochę pracy detektywistycznej i zrozumiesz, co się dzieje.

Krok 1: Sprawdź metodę onCreate () i dodaj rejestrowanie (jogging)

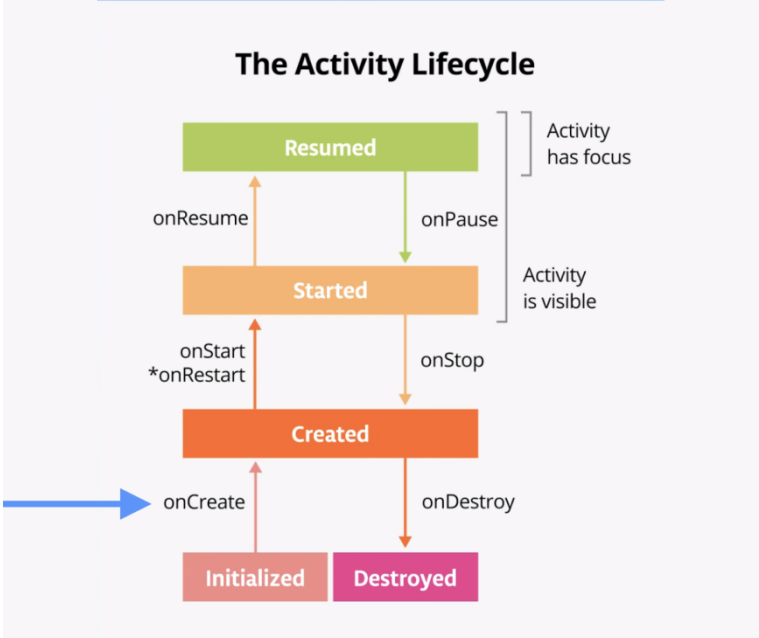
Aby dowiedzieć się, co się dzieje z cyklem życia Androida, warto wiedzieć, kiedy wywoływane są różne metody cyklu życia. Pomoże Ci to wyśledzić, gdzie coś idzie nie tak w DessertClicker.

Prostym sposobem na to jest skorzystanie z interfejsu API rejestrowania systemu Android. Rejestrowanie umożliwia pisanie krótkich wiadomości na konsoli podczas działania aplikacji i można jej użyć, aby pokazać, kiedy uruchamiane są różne wywołania zwrotne (callbacks).

1. Pobierz aplikację startową DessertClicker otwórz ją w Android Studio..
2. Skompiluj i uruchom aplikację, a następnie dotknij kilka razy na obrazie deseru. Zwróć uwagę, jak zmienia się wartość sprzedanych deserów i całkowita kwota w dolarach.
3. Otwórz `MainActivity.kt` i sprawdź metodę `onCreate()` dla tego działania activity

```
override fun onCreate(savedInstanceState: Bundle?) {
...
}
```

Na diagramie cyklu życia activity mogłeś rozpoznać metodę `onCreate()` ponieważ wcześniej używałeś tego wywołania zwrotnego. Jest to jedna metoda, którą każde activity musi wdrożyć. Metoda `onCreate()` to miejsce, w którym należy wykonać jednorazowe inicjalizacje dla swojej aktywności. Na przykład w `onCreate()` nadmuchujesz układ, definiujesz detektory kliknięć (click listeners) lub konfigurujesz powiązanie (data binding).



Metoda cyklu życia `onCreate()` jest wywoływana jeden raz, zaraz po zainicjowaniu działania (gdy nowy obiekt działania jest tworzony w pamięci). Po wykonaniu `onCreate()` działanie uznaje się za utworzone *created*.

Uwaga: Metoda `onCreate()` jest przesłonięciem, dlatego należy natychmiast wywołać funkcję `super.onCreate()`.To samo dotyczy innych metod cyklu życia.

4. W metodzie `onCreate()` tuż po wywołaniu `super.onCreate()`,dodaj następujący wiersz. W razie potrzeby zaimportuj klasę `Log`.

```
Log.i("MainActivity", "onCreate Called")
```

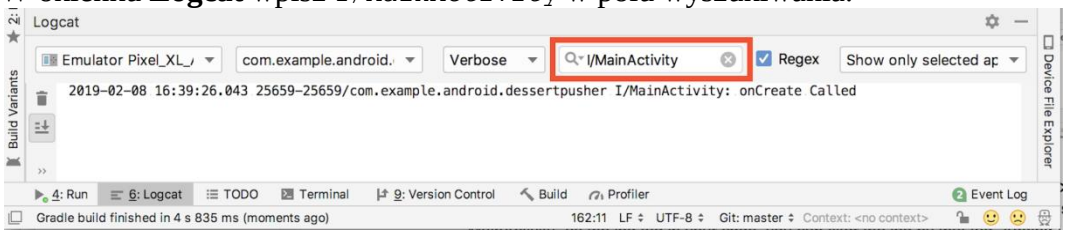
Klasa Log zapisuje komunikaty w Logcat. To polecenie składa się z trzech części:

- *severity* Ważność komunikatu dziennika, to znaczy, jak ważny jest komunikat. W takim przypadku metoda `Log.i()` zapisuje komunikat informacyjny. Inne metody w klasie Log to `Log.e()` w przypadku błędów lub `Log.w()` w przypadku ostrzeżeń.
- Znacznik *tag*, w tym przypadku `"MainActivity"`. Tag jest ciągiem, który pozwala łatwiej znajdować wiadomości w Logcat. Tag jest zazwyczaj nazwą klasy.
- Rzeczywisty komunikat w dzienniku *message*, krótki ciąg znaków, który w tym przypadku to `"onCreate called"`.

5. Skompiluj i uruchom aplikację DessertClicker Po dotknięciu deseru nie widać żadnych różnic w zachowaniu w aplikacji. W Android Studio u dołu ekranu kliknij kartę **Logcat**.

Logcat to konsola do rejestrowania wiadomości. Pojawiają się tutaj wiadomości z Androida o Twojej aplikacji, w tym wiadomości, które jawnie wysyłasz do dziennika za pomocą metody `Log.i()` lub innych metod klasy Log.

6. W okienku **Logcat** wpisz `I/MainActivity` w polu wyszukiwania.



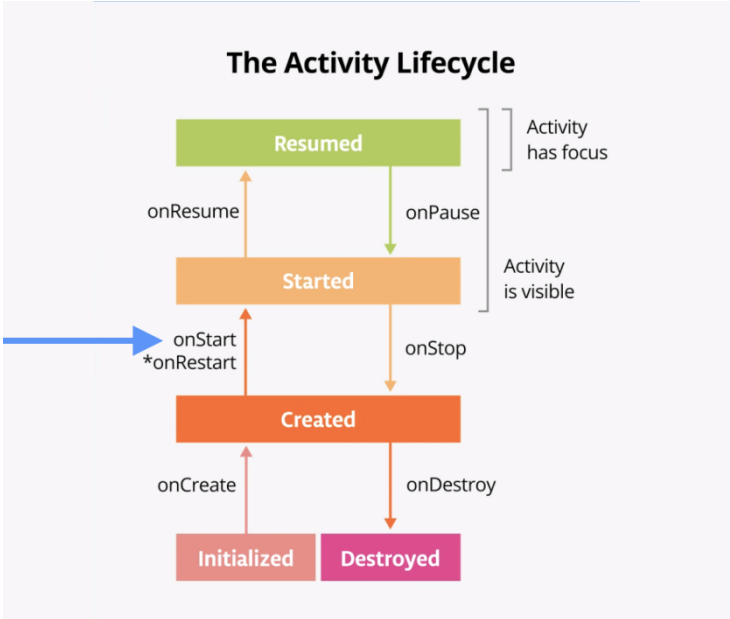
Logcat może zawierać wiele wiadomości, z których większość nie jest dla Ciebie przydatna. Pozycje Logcat można filtrować na wiele sposobów, ale wyszukiwanie jest najłatwiejsze. Ponieważ użyłeś `MainActivity` jako znacznika dziennika w kodzie, możesz użyć tego znacznika do filtrowania dziennika. Dodanie `I/` na początku oznacza, że jest to komunikat informacyjny utworzony przez `Log.i()`.

Komunikat dziennika zawiera datę i godzinę, nazwę pakietu

(`com.example.android.dessertclicker`), tag dziennika (z `I/` na początku), oraz rzeczywisty komunikat. Ponieważ ten komunikat pojawia się w dzienniku, wiesz, że `onCreate()` został wykonany.

Krok 2: Zaimplementuj metodę onStart()

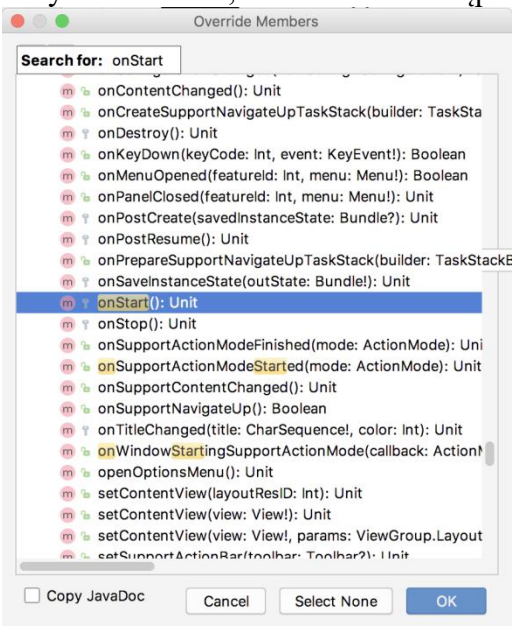
Metoda cyklu życia `onStart()` jest wywoływana tuż po funkcji `onCreate()`.Po uruchomieniu `onStart()` twoja aktywność jest widoczna na ekranie. W przeciwieństwie do funkcji `onCreate()`,która jest wywoływana tylko raz w celu zainicjowania działania, `onStart()` może być wywoływany wiele razy w cyklu życia działania.



Zauważ, że `onStart()` jest sparowany z odpowiednią metodą cyklu życia `onStop()` Jeśli użytkownik uruchomi aplikację, a następnie powróci do ekranu głównego urządzenia, aktywność zostanie zatrzymana i nie będzie już widoczna na ekranie.

1. W Android Studio przy otwartym `MainActivity.kt` wybierz **Code > Override Methods** lub naciśnij `Control+o`. Pojawi się okno dialogowe z ogromną listą

wszystkich metod, które można zastąpić w tej klasie.



2. Zacznij wpisywać `onStart` aby wyszukać właściwą metodę. Aby przewinąć do następnego pasującego elementu, użyj strzałki w dół. Wybierz z listy opcję `onStart()` i kliknij przycisk OK, aby wstawić kod zastępujący (override code). Kod wygląda następująco:

```
override fun onStart() {
    super.onStart()
}
```

Wskazówka: Android Studio wstawia zastąpiony kod metody w następnym dostępnym odpowiednim miejscu w klasie. Jeśli chcesz umieścić przesłonięcia cyklu życia w określonym miejscu (np. Na końcu klasy), ustaw punkt wstawiania przed użyciem **Override Methods**.

3. W metodzie `onStart()` dodaj komunikat w dzienniku:

```
override fun onStart() {
    super.onStart()

    Log.i("MainActivity", "onStart Called")
}
```

4. Skompiluj i uruchom aplikację DessertClicker i otwórz okienko Logcat. Wpisz `I/MainActivity` w polu wyszukiwania, aby przefiltrować dziennik. Zauważ, że zarówno metody `onCreate()`, jak i `onStart()` były wywoływane jedna po drugiej i że twoja aktywność jest widoczna na ekranie.
5. Naciśnij przycisk Home button na urządzeniu, a następnie użyj recents screen aby powrócić do aktywności. Zauważ, że działanie wznawia się tam, gdzie zostało przerwane, z tymi samymi wartościami oraz że `onStart()` jest logowany po raz drugi w Logcat. Zauważ również, że metoda `onCreate()` zwykle nie jest ponownie wywoływana.

Uwaga: eksperymentując z urządzeniem i obserwując wywołania zwrotne cyklu życia, możesz zauważyć nietypowe zachowanie podczas obracania urządzenia. O tym zachowaniu dowiesz się w następnym ćwiczeniu.

Zadanie: Użyj Timber do logowania

W tym zadaniu zmodyfikujesz aplikację, aby korzystała z popularnej biblioteki rejestrowania o nazwie `Timber`. `Timber` ma kilka zalet w porównaniu do wbudowanej klasy Android Log. W szczególności biblioteka `Timber`:

- Generuje dla Ciebie tag dziennika na podstawie nazwy klasy.
- Pomaga uniknąć wyświetlania dzienników w wydanej wersji aplikacji na Androida.
- Umożliwia integrację z bibliotekami zgłaszającymi awarie.

Natychmiast zobaczysz pierwszą korzyść; inne, docenisz, tworząc większe aplikacje.

Krok 1: Dodaj Timber do Gradle

1. Odwiedź ten link do projektu Timber project on GitHub, i skopiuj wiersz kodu pod nagłówkiem **Download** który zaczyna się od słowa `implementation`. Wiersz kodu będzie wyglądał mniej więcej tak, chociaż numer wersji może być inny.

```
implementation 'com.jakewharton.timber:timber:4.7.1'
```

2. W Android Studio, w widoku Projekt: Android, rozwiń **Gradle Scripts** i otwórz plik **build.gradle (Module: app)** .
3. W sekcji zależności wklej skopiowany wiersz kodu.

```
dependencies {
    ...
    implementation 'com.jakewharton.timber:timber:4.7.1'
}
```

4. Kliknij **Sync Now** w prawym górnym rogu Android Studio. Kompilacja powinna zostać wykonana bez błędów.

Krok 2: Utwórz klasę aplikacji i zainicjuj Timber

W tym kroku tworzysz klasę `Application`. `Application` jest klasą podstawową, która zawiera globalny stan aplikacji dla całej aplikacjiJest to także główny obiekt używany przez system operacyjny do interakcji z aplikacją. Istnieje domyślna klasa `Application`, której Android używa, jeśli jej nie określisz, więc zawsze jest tworzony obiekt `Application` dla twojej aplikacji, bez potrzeby robienia czegokolwiek specjalnego, aby ją utworzyć.

`Timber` korzysta z klasy `Application` ponieważ cała aplikacja będzie korzystać z tej biblioteki rejestrowania, a bibliotekę należy zainicjować raz, zanim wszystko inne zostanie skonfigurowane. W takich przypadkach można „podklasowa” klasę `Application` i zastępować wartości domyślne za pomocą własnej niestandardowej implementacji..

Warning: dodawanie własnego kodu do klasy `Application` może być kuszące, ponieważ klasa jest tworzona przed wszystkimi Twoimi activity i może utrzymywać stan globalny. Ale ponieważ bardzo podatne na błędy jest tworzenie czytelnych i zapisywalnych zmiennych

statycznych, które są globalnie dostępne, łatwo jest nadużywać klasy `Application`. Unikaj umieszczania kodu aktywności w klasie `Application`, chyba że kod jest naprawdę potrzebny.

Po utworzeniu klasy `Application` musisz określić klasę w manifeście systemu Android.

1. W pakiecie `dessertclicker` utwórz nową klasę Kotlin o nazwie `ClickerApplication`. Aby to zrobić, rozwiń **app > java** i kliknij prawym przyciskiem myszy **com.example.android.dessertclicker**. Wybierz **New > Kotlin File/Class**.
2. Nazwij klasę **ClickerApplication** i ustaw **Kind** na **Class**. Kliknij **OK**.

Android Studio tworzy nową klasę `ClickerApplication` i otwiera ją w edytorze kodu. Kod wygląda następująco:

```
package com.example.android.dessertclicker

class ClickerApplication {
}
```

3. Zmień definicję klasy na podklasę `Application` i zaimportuj klasę `Application` jeśli to konieczne.

```
class ClickerApplication : Application() {
```

4. Aby zastąpić metodę `onCreate()` wybierz **Code > Override Methods** ub naciśnij `Control+o`.

```
class ClickerApplication : Application() {
   override fun onCreate() {
       super.onCreate()
   }
}
```

5. Wewnątrz tej metody `onCreate()` zainicjuj bibliotekę `Timber`:

```
override fun onCreate() {
    super.onCreate()

    Timber.plant(Timber.DebugTree())
}
```

Ten wiersz kodu inicjuje bibliotekę `Timber` dla Twojej aplikacji, dzięki czemu możesz korzystać z biblioteki w swoich activities.

6. Otwórz **AndroidManifest.xml**.
7. W górnej części elementu `<application>` dodaj nowy atrybut dla klasy `ClickerApplication` , aby Android wiedział, że używa klasy `Application` zamiast domyślnej.

```
<application
   android:name=".ClickerApplication"
...
```

Note: jeśli nie dodasz niestandardowej klasy aplikacji do manifestu Androida, aplikacja będzie działać bez błędów. Jednak aplikacja nie będzie korzystała z Twojej klasy i nigdy nie zobaczysz żadnych informacji logowania z `Timber`.

Krok 3: Dodaj instrukcje dziennika Timber

W tym kroku zmieniasz wywołania `Log.i()` aby korzystać z `Timber`, a następnie wdrażasz rejestrowanie dla wszystkich innych metod cyklu życia.

1. Otwórz `MainActivity` i przewiń do `onCreate()`.Zamień `Log.i()` z `Timber.i()` usuń tag dziennika.

```
Timber.i("onCreate called")
```

Podobnie jak klasa `Log`, `Timber` również używa metody `i()` do komunikatów informacyjnych. Zauważ, że w `Timber` nie musisz dodawać tagu dziennika (log tag); `Timber` automatycznie używa nazwy klasy jako znacznika dziennika.

2. Podobnie zmień wywołanie Log w `onStart()`:

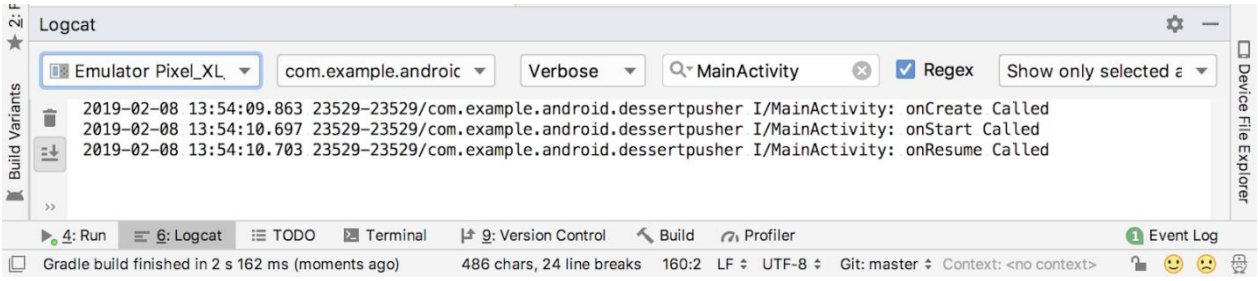
```
override fun onStart() {
   super.onStart()

   Timber.i("onStart Called")
}
```

3. Skompiluj i uruchom aplikację DessertClicker app, i otwórz Logcat. Zauważ, że nadal widzisz te same komunikaty dziennika dla `onCreate()` i `onStart()`,tylko teraz `Timber` generuje te komunikaty, a nie klasa `Log`.
4. Override Zastąp pozostałe metody cyklu życia w `MainActivity`, i dodaj dla każdego z nich instrukcje `Timber` Oto kod:

```
override fun onResume() {
   super.onResume()
   Timber.i("onResume Called")
}

override fun onPause() {
   super.onPause()
   Timber.i("onPause Called")
}

override fun onStop() {
   super.onStop()
   Timber.i("onStop Called")
}

override fun onDestroy() {
   super.onDestroy()
   Timber.i("onDestroy Called")
}

override fun onRestart() {
   super.onRestart()
   Timber.i("onRestart Called")
```

```
}
```

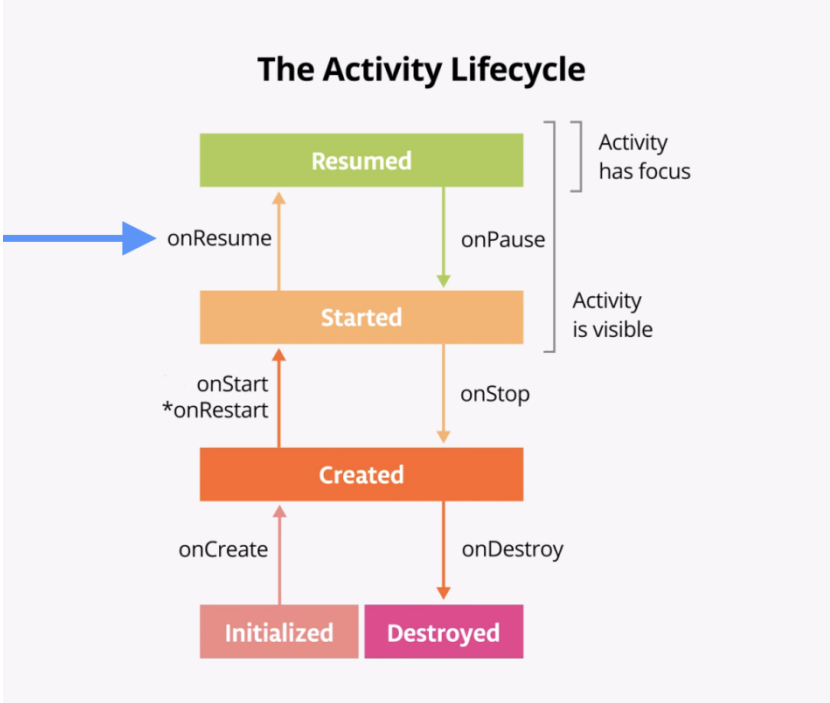
5. Skompiluj i ponownie uruchom DessertClicker i sprawdź Logcat. Tym razem zauważ, że oprócz `onCreate()` i `onStart()`,pojawia się komunikat dziennika dla wywołania zwrotnego cyklu życia `onResume()`.



Gdy działanie rozpoczyna się od zera, wszystkie trzy wywołania zwrotne cyklu życia są wywoływane w kolejności:

- `onCreate()` aby utworzyć aplikację.
- `onStart()` aby go uruchomić i uczynić widocznym na ekranie.
- `onResume()` aby skoncentrować się na aktywności i przygotować ją do interakcji z użytkownikiem.

Pomimo nazwy metoda `onResume()` jest wywoływana podczas uruchamiania, nawet jeśli nie ma nic do wznowienia.



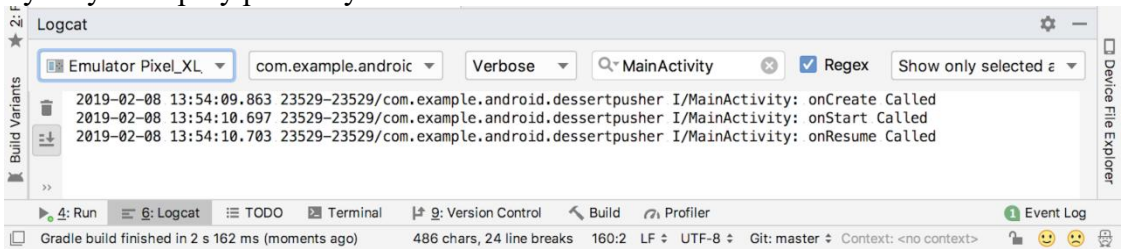
5. Zadanie: poznaj przypadki użycia (use cases) w cyklu życia

Teraz, gdy aplikacja DessertClicker jest skonfigurowana do rejestrowania, możesz rozpocząć korzystanie z aplikacji na różne sposoby i zbadać, w jaki sposób wyzwalane są wywołania zwrotne cyklu życia.

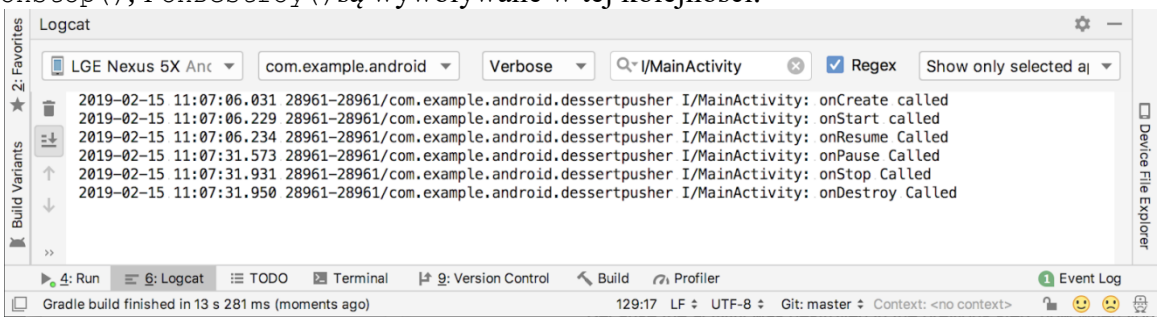
Use case 1: Otwieranie i zamykanie działania

Zaczynasz od najbardziej podstawowego przypadku użycia, którym jest uruchomienie aplikacji po raz pierwszy, a następnie całkowite zamknięcie aplikacji.

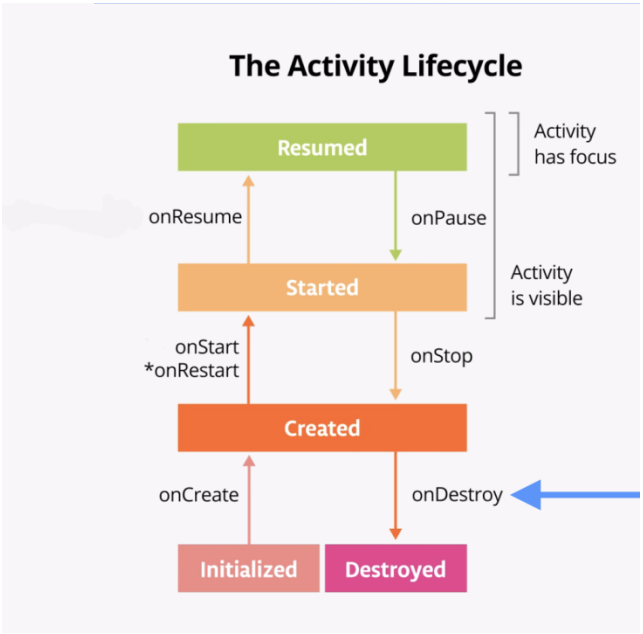
1. Skompiluj i uruchom aplikację DessertClicker, jeśli jeszcze nie jest uruchomiona. Jak widzieliście, wywołania zwrotne `onCreate()`, `onStart()`, i `onResume()` są wywoływane przy pierwszym uruchomieniu działania.



2. Kliknij cupcake kilka razy.
3. Kliknij Back button przycisk Wstecz na urządzeniu. Zauważ w Logcat, że `onPause()`, `onStop()`, i `onDestroy()` są wywoływane w tej kolejności.

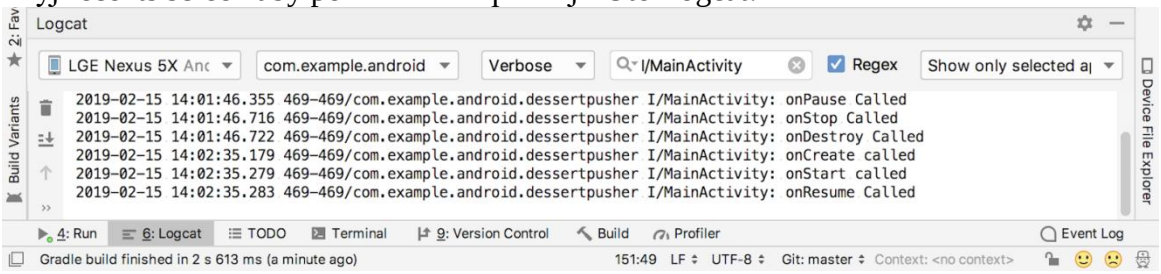


W takim przypadku użycie przycisku Wstecz powoduje całkowite zamknięcie działania (i aplikacji). Wykonanie metody `onDestroy()` oznacza, że działanie zostało całkowicie zamknięte i można je wyrzucić (garbage-collected) Odśmiecanie Garbage collection odnosi się do automatycznego czyszczenia obiektów, których już nie będziesz używać. Po wywołaniu funkcji `onDestroy()` system operacyjny wie, że te zasoby są usuwalne, i rozpoczyna czyszczenie tej pamięci.



Twoja aktywność może zostać całkowicie zamknięta również, jeśli kod ręcznie wywoła metodę `finish()` lub użytkownik wymusi zamknięcie aplikacji. (Na przykład użytkownik może wymusić zamknięcie aplikacji na ostatnim ekranie, klikając X w rogu okna.) System Android może również samodzielnie zamknąć Twoją aktywność, jeśli aplikacja nie była wyświetlana na ekranie przez długi czas. Android ma tu na celu oszczędzanie baterii i umożliwienie korzystania z zasobów aplikacji przez inne aplikacje.

4. Użyj recents screen aby powrócić do aplikacji. Oto Logcat:



Aktywność została zniszczona w poprzednim kroku, więc po powrocie do aplikacji Android uruchamia nową aktywność i wywołuje metody `onCreate()`, `onStart()`, i `onResume()` Zauważ, że nie zachowano żadnej statystyki DessertClicker z poprzedniego działania.

Kluczową kwestią jest to, że `onCreate()` i `onDestroy()` są wywoływane tylko raz w czasie istnienia pojedynczej instancji działania: `onCreate()` w celu zainicjowania aplikacji po raz pierwszy, a `onDestroy()` w celu oczyszczenia zasobów używanych przez twoją aplikację.

Metoda `onCreate()` jest ważnym krokiem; tutaj odbywa się cała Twoja pierwsza inicjalizacja, gdzie konfigurujesz układ po raz pierwszy poprzez nadmuchanie go i gdzie inicjujesz swoje zmienne.

Use case 2: Nawigowanie poza activity i powrót do niego

Teraz, gdy uruchomiłeś aplikację i całkowicie ją zamknąłeś, widziałeś większość stanów cyklu życia, kiedy działanie zostało utworzone po raz pierwszy. Widziałeś także wszystkie stany cyklu życia, przez które aktywność przechodzi, kiedy zostaje całkowicie zamknięta i zniszczona. Ale użytkownicy wchodzą również w interakcje z urządzeniami z systemem Android, przełączają się między aplikacjami, wracają do ekranu głównego, uruchamiają nowe aplikacje, radzą sobie z przerwami spowodowanymi innymi czynnościami, takimi jak rozmowy telefoniczne.

Twoja aktywność nie zamyka się całkowicie za każdym razem, gdy użytkownik ją opuści:

- Gdy twoja aktywność nie jest już widoczna na ekranie, nazywa się to umieszczaniem aktywności w tle. *background*. (Przeciwnie, gdy aktywność znajduje się na pierwszym planie lub na ekranie *foreground*,)
- Gdy użytkownik powróci do aplikacji, ta sama aktywność zostanie ponownie uruchomiona i znów będzie widoczna. Ta część cyklu życia nazywa się widocznym cyklem życia aplikacji. *visible* lifecycle.

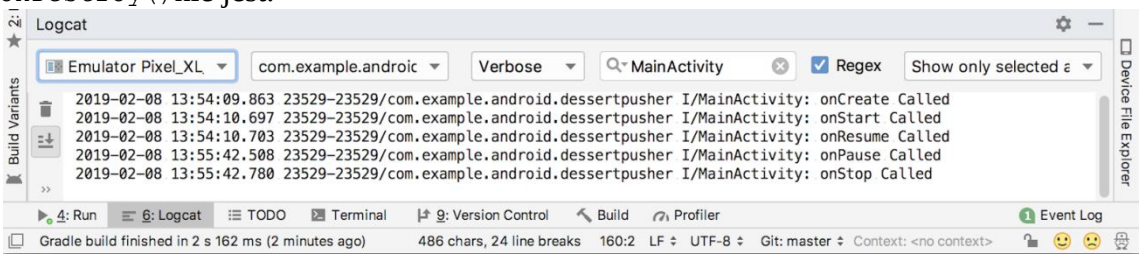
Gdy aplikacja działa w tle, nie powinna być aktywnie uruchomiona, aby zachować zasoby systemowe i żywotność baterii. Musimy korzystać z cyklu życia activity i jego wywołań zwrotnych, aby wiedzieć, kiedy aplikacja przenosi się w tło, dzięki czemu możesz wstrzymać wszelkie bieżące operacje. Następnie ponownie uruchomić operacje, gdy aplikacja pojawi się na pierwszym planie.

Na przykład rozważ aplikację, która uruchamia symulację fizyki. Aplikacja wymaga wielu obliczeń, które bardzo obciążają procesor urządzenia. Jeśli połączenie telefoniczne przerwie symulację, użytkownik może być niezadowolony kiedy po powrocie do aplikacji aby sprawdzić, czy symulacja została zakończona zorientuje się że została przerwana.

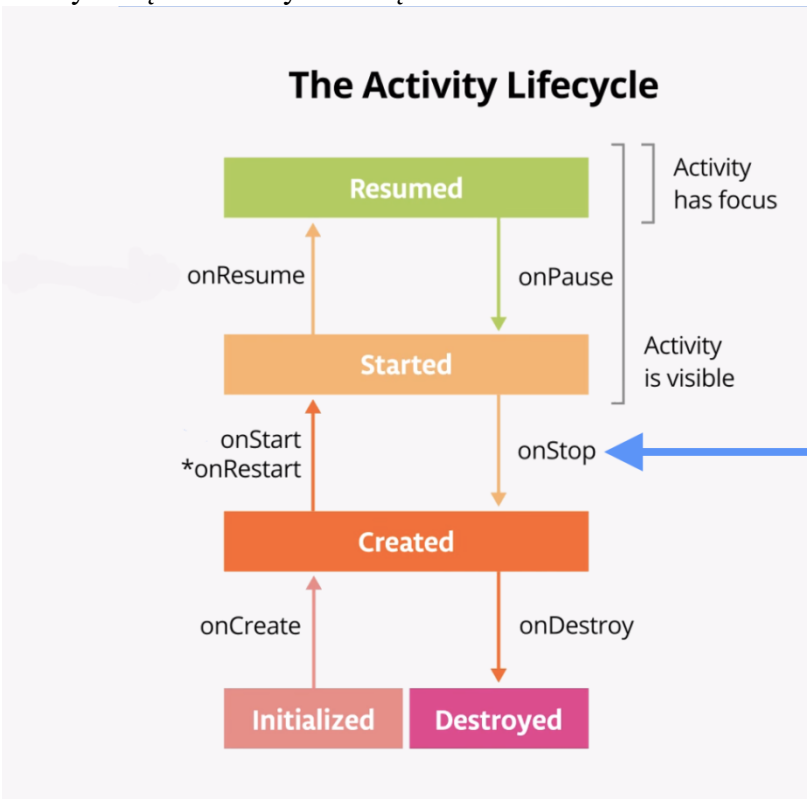
Ważna jest również kwestia wydajności. Załóżmy, że użytkownik otworzył 20 aplikacji korzystających z symulacji fizycznych mocno obciążających procesor. Jeśli działania tych aplikacji nie są wyświetlane na ekranie, ale nadal wykonują ciężkie obliczenia renderowania w tle, spowolni to działanie całego telefonu. W takiej sytuacji system zamknie aplikacje bez informowania o tym użytkownika.

W tym kroku zbadamy cykl życia aktywności, gdy aplikacja przechodzi w tło i wraca na pierwszy plan.

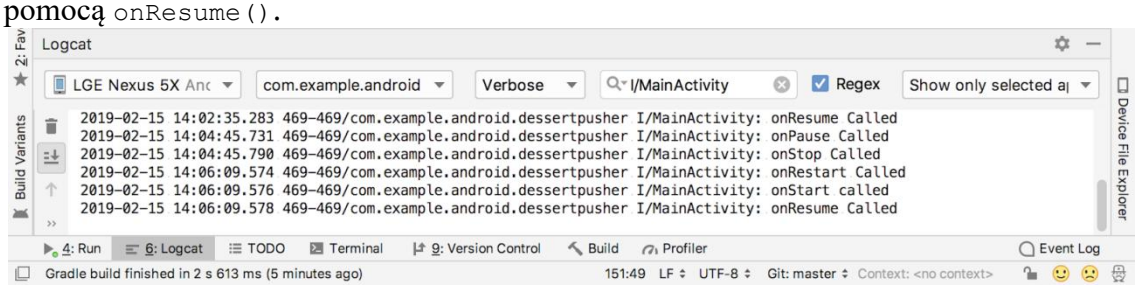
1. Przy uruchomionej aplikacji DessertClicker kliknij kilka razy cupcake .
2. Naciśnij przycisk Home na swoim urządzeniu i obserwuj Logcat w Android Studio. Powrót do ekranu głównego powoduje wyłączenie aplikacji w tle, a nie jej całkowite wyłączenie. Zauważ, że wywoływane są metody `onPause()` i `onStop()` ale `onDestroy()` nie jest.



Po wywołaniu funkcji `onPause()` aplikacja nie jest już aktywna. Po funkcji `onStop()`,aplikacja nie jest już widoczna na ekranie. Chociaż aktywność została zatrzymana, obiekt `Activity` jest nadal w pamięci, w tle. Aktywność nie została zniszczona. Użytkownik może powrócić do aplikacji, więc system Android utrzymuje zasoby związane z aktywnością.



3. Użyj recents screen , aby powrócić do aplikacji. Zauważ w Logcat, że aktywność jest restartowana za pomocą `onRestart()` i `onStart()`,a następnie wznawiana za pomocą `onResume()`.



Gdy aktywność powraca na pierwszy plan, metoda `onCreate()` nie jest wywoływana ponownie. Obiekt aktywności nie został zniszczony, więc nie trzeba go ponownie tworzyć. Zamiast `onCreate()`,wywoływana jest metoda `onRestart()` Zauważ, że tym razem, gdy aktywność powróci na pierwszy plan, liczba sprzedanych deserów zostaje zachowana.

4. Uruchom co najmniej jedną aplikację inną niż DessertClicker, aby na recents screen urządzenia było kilka aplikacji.
5. Wyświetl recents screen i otwórz inną ostatnią aktywność. Następnie wróć do recent apps i przenieś DessertClicker na pierwszy plan.

Zauważ, że w Logcat widzisz te same wywołania zwrotne, co po naciśnięciu przycisku Początek. `onPause()` i `onStop()` są wywoływane, gdy aplikacja przechodzi w tło, a następnie `onRestart()`, `onStart()`, i `onResume()` gdy powraca.

Ważną kwestią jest to, że `onStart()` i `onStop()` są wywoływane wiele razy, gdy użytkownik nawiguje do i z działania. Powinieneś zastąpić te metody, aby zatrzymać aplikację, gdy przejdzie ona w tło, lub uruchom ją ponownie, gdy wróci na pierwszy plan.

A co z `onRestart()`?Metoda `onRestart()` jest bardzo podobna do metody `onCreate()`. Zarówno `onCreate()` jak i `onRestart()` jest wywoływana, zanim aktywność stanie się widoczna. Metoda `onCreate()` jest wywoływana tylko za pierwszym razem, a następnie wywoływana jest metoda `onRestart()` Metoda `onRestart()` to miejsce do umieszczenia kodu, który chcesz wywołać tylko wtedy, gdy twoja aktywność nie jest uruchamiana po raz pierwszy.

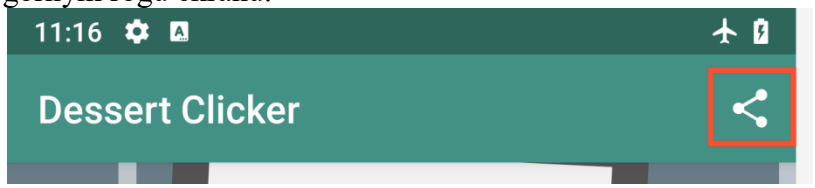
Use case 3: Częściowo ukryte activity

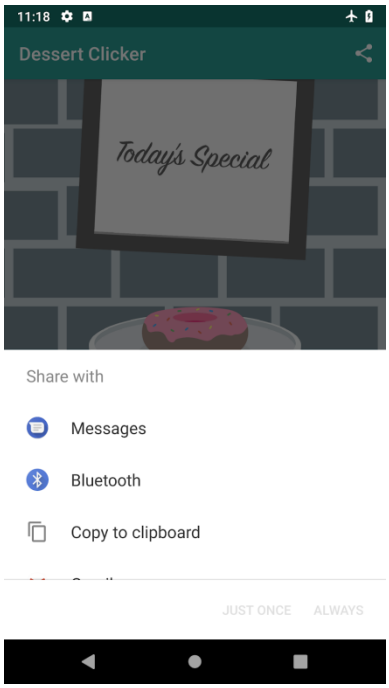
Dowiedziałeś się, że po uruchomieniu aplikacji i wywołaniu funkcji `onStart()` aplikacja staje się widoczna na ekranie. Po wznowieniu aplikacji i wywołaniu funkcji `onResume()` aplikacja jest w interakcji z użytkownikiem. Część cyklu życia, w której aplikacja jest w pełni wyświetlana na ekranie i skupia się na użytkowniku, nazywa się interaktywnym cyklem życia. *interactive* lifecycle.

Gdy aplikacja przechodzi w tło, fokus jest tracony po `onPause()`,a aplikacja nie jest już widoczna po `onStop()`.

Różnica między skupieniem a widocznością jest ważna, ponieważ możliwe jest, że activity będzie częściowo widoczne na ekranie, ale nie będzie miało skupienia użytkownika. W tym kroku sprawdzimy jeden przypadek, w którym aktywność jest częściowo widoczna, ale nie koncentruje się na użytkowniku.

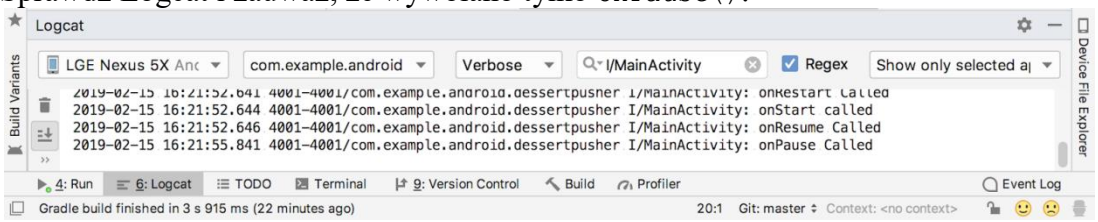
1. Przy uruchomionej aplikacji DessertClicker kliknij przycisk Udostępnij w prawym górnym rogu ekranu.





Aktywność udostępniania pojawia się w dolnej połowie ekranu, ale aktywność jest nadal widoczna w górnej połowie.

2. Sprawdź Logcat i zauważ, że wywołano tylko `onPause()`.



W tym przypadku funkcja `onStop()` nie jest wywoływana, ponieważ aktywność jest nadal częściowo widoczna. Ale działanie nie koncentruje się na użytkowniku i użytkownik nie może z nim współdziałać. Aktywność „udostępniania" na pierwszym planie koncentruje się na użytkownikach.

Dlaczego ta różnica jest ważna? Zastanów się nad aplikacją fizyki. Możesz chcieć zatrzymać symulację, gdy aplikacja jest w tle, i kontynuować działanie, gdy aplikacja jest częściowo zasłonięta. W takim przypadku zatrzymasz symulację w `onStop()`.Jeśli chcesz, aby symulacja zatrzymała się, gdy działanie jest częściowo zasłonięte, wstaw kod, aby zatrzymać symulację w `onPause()`.

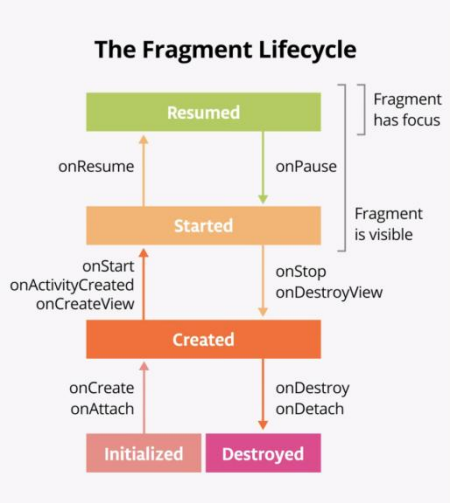
Jakikolwiek kod wykonując się w`onPause()` blokuje wyświetlanie innych rzeczy, więc zachowaj „lekki kod" w `onPause()` Na przykład, jeśli przychodzi połączenie telefoniczne, kod w `onPause()` może opóźnić powiadomienie o połączeniu przychodzącym.

3. Kliknij poza oknem dialogowym udostępniania, aby powrócić do aplikacji i zauważ, że wywoływana jest funkcja `onResume()`.

Zarówno `onResume()`, jak i `onPause()` mają do czynienia z fokusem. Metoda `onResume()` jest wywoływana, gdy aktywność ma fokus, a `onPause()` jest wywoływana, gdy aktywność traci fokus.

6. Zadanie: badanie cyklu życia fragmentu

Cykl życia fragmentu Androida jest podobny do cyklu życia aktywności wzbogacony o kilka metod specyficznych dla fragmentu.



W tym zadaniu zbadamy aplikację AndroidTrivia, którą zbudowałeś w poprzednich ćwiczeniach.

Każdy ekran w aplikacji AndroidTrivia jest `Fragment`.

Aby uprościć sprawę, w tym zadaniu korzystasz z interfejsu API rejestrowania systemu Android, a nie z biblioteki.

1. Otwórz aplikację AndroidTrivia.
2. Otwórz plik `TitleFragment.kt`.
3. Przewiń w dół do metody `onCreateView()`.
4. Dodaj instrukcję rejestrowania do metody `onCreateView()` między wierszem `setHasOptionsMenu()` a ostatnim wywołaniem return:

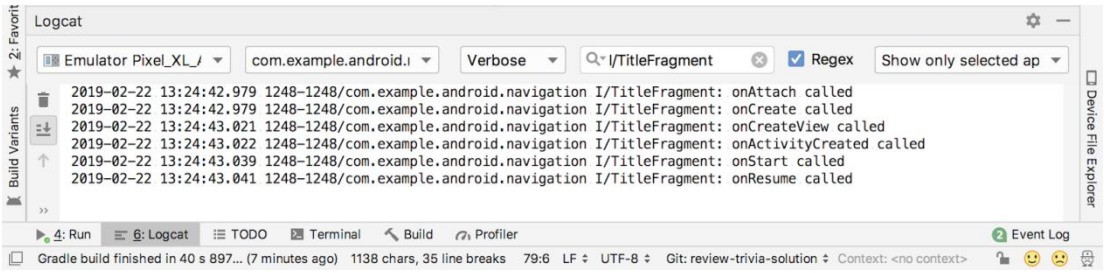
```
setHasOptionsMenu(true)

Log.i("TitleFragment", "onCreateView called")

return binding.root
```

5. Tuż poniżej metody `onCreateView()` dodaj instrukcje rejestrowania dla każdej z pozostałych metod cyklu życia fragmentu. Oto kod:

```
override fun onAttach(context: Context?) {
    super.onAttach(context)
    Log.i("TitleFragment", "onAttach called")
}
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.i("TitleFragment", "onCreate called")
}
override fun onActivityCreated(savedInstanceState: Bundle?) {
```

```
    super.onActivityCreated(savedInstanceState)
    Log.i("TitleFragment", "onActivityCreated called")
}
override fun onStart() {
    super.onStart()
    Log.i("TitleFragment", "onStart called")
}
override fun onResume() {
    super.onResume()
    Log.i("TitleFragment", "onResume called")
}
override fun onPause() {
    super.onPause()
    Log.i("TitleFragment", "onPause called")
}
override fun onStop() {
    super.onStop()
    Log.i("TitleFragment", "onStop called")
}
override fun onDestroyView() {
    super.onDestroyView()
    Log.i("TitleFragment", "onDestroyView called")
}
override fun onDetach() {
    super.onDetach()
    Log.i("TitleFragment", "onDetach called")
}
```

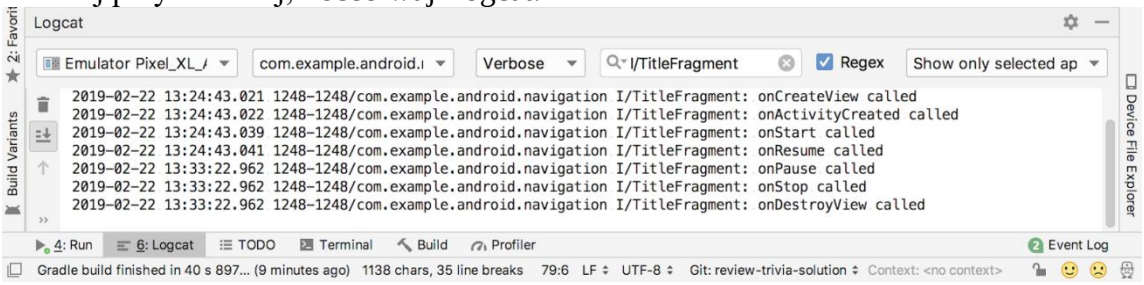
6. Skompiluj i uruchom aplikację, a następnie otwórz Logcat.
7. Wpisz `I/TitleFragment` w polu wyszukiwania, aby przefiltrować log. Po uruchomieniu aplikacji Logcat będzie wyglądał podobnie do następującego zrzutu ekranu:



Tutaj możesz zobaczyć cały cykl życia fragmentu, w tym te wywołania zwrotne:

- `onAttach()`:Wywoływany, gdy fragment jest powiązany z aktywnością właściciela.
- `onCreate()`:Podobnie jak `onCreate()` dla activity, `onCreate()` dla fragmentu jest wywoływany w celu początkowego utworzenia fragmentu (innego niż f layout).
- `onCreateView()`:Wywoływany w celu nadmuchania układu fragmentu.
- `onActivityCreated()`:Wywoływany, gdy czynność `onCreate()` aktywności właściciela zostanie zakończona. Twój fragment nie będzie mógł uzyskać dostępu do activity dopóki ta metoda nie zostanie wywołana.
- `onStart()`:Wywoływany, gdy fragment stanie się widoczny; równolegle do activity `onStart()`.
- `onResume()`:Wywoływany, gdy fragment zyskuje skupienie użytkownika; równolegle do activity `onResume()`.

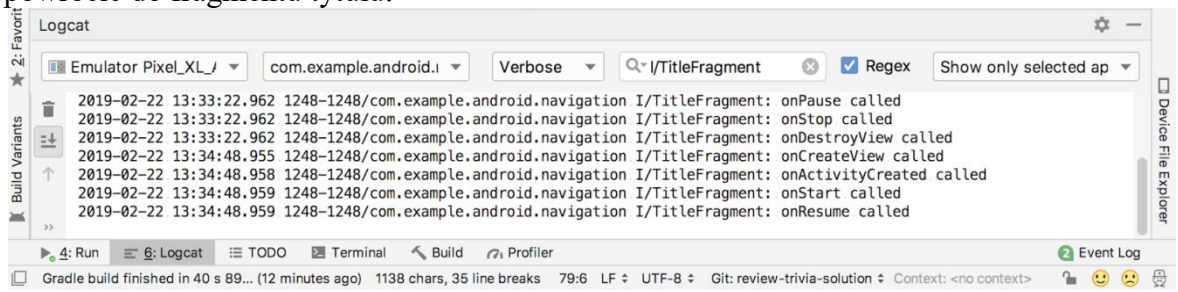
8. Naciśnij przycisk Graj, i obserwuj Logcat.



Otwarcie następnego fragmentu powoduje zamknięcie fragmentu tytułowego i wywołanie tych metod cyklu życia:

- `onPause()`:Wywoływany, gdy fragment traci fokus użytkownika; równolegle do activity `onPause()`.
- `onStop()`:Wywoływany, gdy fragment nie jest już widoczny na ekranie; równolegle do activity `onStop()`.
- `onDestroyView()`:Wywoływany, gdy widok fragmentu nie jest już potrzebny, aby oczyścić zasoby powiązane z tym widokiem.

9. W aplikacji dotknij przycisku W górę (strzałka w lewym górnym rogu ekranu), aby powrócić do fragmentu tytułu.



Tym razem, `onAttach()` i `onCreate()` prawdopodobnie nie są wywoływane w celu uruchomienia fragmentu. Obiekt fragmentu nadal istnieje i jest nadal dołączony do aktywności właściciela, więc cykl życia rozpoczyna się ponownie od `onCreateView()`.

10. Naciśnij przycisk Home urządzenia. Zauważ w Logcat, że wywoływane są tylko `onPause()` i `onStop()` are called. Jest to to samo zachowanie, co w przypadku activity: powrót do ekranu startowego powoduje umieszczenie activity i fragmentu w tle.
11. Użyj recents screen aby wrócić do aplikacji. Podobnie jak w przypadku activity, , wywoływane są metody `onStart()` i `onResume()` w celu przywrócenia fragmentu na pierwszy plan.