3. Zadanie: Unikanie błędów cyklu życia

W tym zadaniu zapoznasz się z bardziej złożonym przykładem zarządzania zadaniami cyklu życia w aplikacji DessertClicker.

Step 1: Skonfiguruj DeserTimer

1. Otwórz aplikację DessertClicker
2. W widoku **Project** rozwiń j**ava > com.example.android.dessertclicker** i otwórz `DessertTimer.kt`. Zauważ, że teraz cały kod jest „zakomentowany", więc nie działa jako część aplikacji.
3. Wybierz cały kod w oknie edytora. Wybierz **Code > Comment with Line Comment**, lub naciśnij `Control+/` . To polecenie usuwa komentarz z kodu całego pliku. (Android Studio może wyświetlać nierozwiązane błędy referencyjne, dopóki nie przebudujesz aplikacji).
4. Zauważ, że klasa `DessertTimer` obejmuje `startTimer()` i `stopTimer()`,które uruchamiają i zatrzymują stoper. Gdy `startTimer()` jest uruchomiony, zegar drukuje komunikat dziennika co sekundę, z całkowitą liczbą sekund, które upłynęły. Z kolei metoda `stopTimer()` zatrzymuje licznik czasu i instrukcje dziennika.

Note: Klasa `DessertTimer` używa wątku tła (background thread) dla timera z powiązanymi klasami `Runnable` i `Handler`. Nie musisz wiedzieć jak to działa w tym ćwiczeniu (dowiesz się więcej o wątkach w późniejszych ćwiczeniach).

5. Otwórz `MainActivity.kt`. Na górze klasy, tuż pod zmienną `dessertsSold` dodaj zmienną dla timera:

```
private lateinit var dessertTimer : DessertTimer;
```

6. Przewiń w dół do `onCreate()` i utwórz nowy obiekt `DessertTimer` zaraz po wywołaniu `setOnClickListener()`:

```
dessertTimer = DessertTimer()
```

Teraz, gdy masz obiekt dessert timer object, zastanów się, gdzie powinieneś zacząć i zatrzymać stoper, aby działał tylko wtedy, gdy aktywność jest wyświetlana na ekranie. W następnych krokach przyjrzymy się kilku opcjom.

Step 2: Uruchom i zatrzymaj stoper

Metoda `onStart()` est wywoływana tuż przed tym, jak aktywność stanie się widoczna. Metoda `onStop()` jest wywoływana, gdy aktywność przestaje być widoczna. Te wywołania zwrotne wydają się być dobrymi kandydatami do uruchomienia i zatrzymania stopera.

1. W klasie `MainActivity` uruchom licznik czasu w wywołaniu zwrotnym `onStart()`:

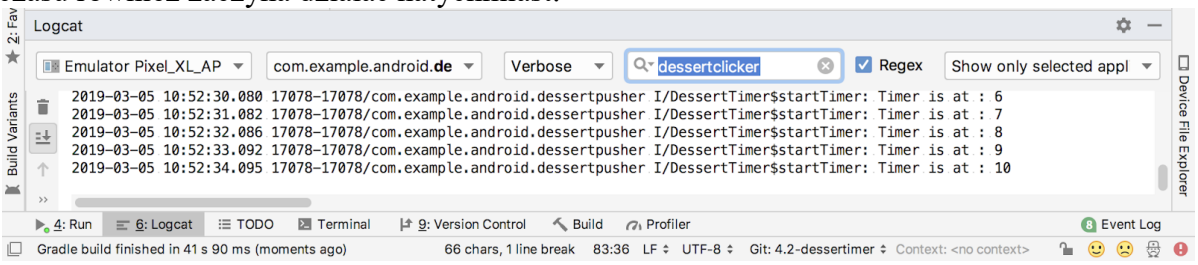
```
override fun onStart() {
   super.onStart()
```

```
    dessertTimer.startTimer()

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

2. Zatrzymaj stoper w `onStop()`:

```
override fun onStop() {
    super.onStop()
    dessertTimer.stopTimer()

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

3. Skompiluj i uruchom aplikację. W Android Studio kliknij okienko Logcat. W polu wyszukiwania Logcat wpisz `dessertclicker`, , który będzie filtrował według klas `MainActivity` i `DessertTimer` Zwróć uwagę, że po uruchomieniu aplikacji licznik czasu również zaczyna działać natychmiast.



4. Kliknij przycisk Wstecz i zauważ, że stoper ponownie się zatrzymuje. Timer zatrzymuje się, ponieważ zarówno activity jak i kontrolowany przez niego zegar zostały zniszczone.
5. Użyj recents screen , aby powrócić do aplikacji. Zauważ w Logcat, że licznik czasu restartuje się od 0.
6. Kliknij przycisk Udostępnij. Zauważ w Logcat, że stoper nadal działa.

7. Kliknij przycisk **Home** button. Zauważ w Logcat, że stoper przestaje działać.
8. Użyj recents screen , aby powrócić do aplikacji. Zauważ w Logcat, że minutnik uruchamia się ponownie od miejsca, w którym został przerwany.
9. W `MainActivity`, w metodzie `onStop()` zakomentuj wywołanie `stopTimer()`. Zakomentowanie `stopTimer()` pokazuje przypadek, w którym zaczynasz operację w `onStart()`,ale zapominasz zatrzymać ją ponownie w `onStop()`.
10. Skompiluj i uruchom aplikację, a po uruchomieniu stopera kliknij przycisk Home Mimo że aplikacja działa w tle, timer działa i stale korzysta z zasobów systemowych. Utrzymywanie działania timera jest wyciekiem pamięci dla aplikacji i prawdopodobnie nie jest to pożądane zachowanie.

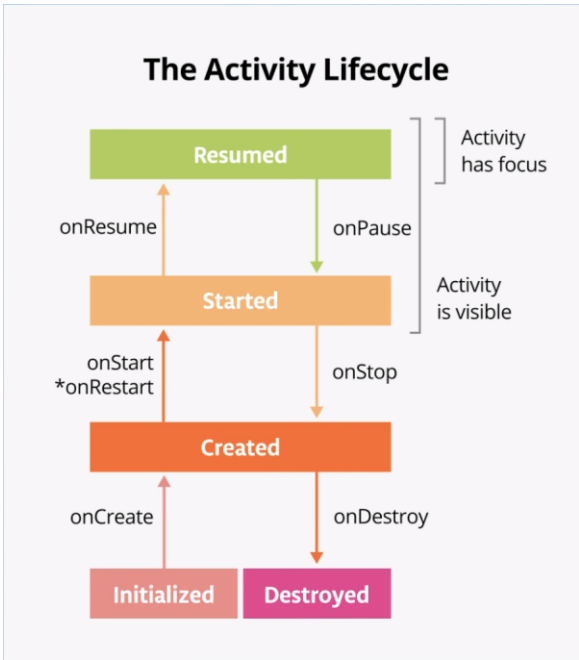
    Ogólny wzorzec jest taki, że kiedy konfigurujesz lub uruchamiasz coś w funkcji callback, zatrzymujesz lub usuwasz to w odpowiedniej funkcji callback. W ten sposób unikasz uruchamiania czegokolwiek, gdy nie jest już potrzebne.

Uwaga: w niektórych przypadkach, takich jak odtwarzanie muzyki, chcesz utrzymać działanie. Istnieją odpowiednie i skuteczne sposoby na utrzymanie działania, ale wykraczają one poza zakres tej lekcji.

11. Odkomentuj linię w `onStop()` w której zatrzymałeś stoper.
12. Wytnij i wklej wywołanie `startTimer()` z `onStart()` do `onCreate()`.Ta zmiana pokazuje przypadek, w którym zarówno inicjujesz, jak i uruchamiasz zasób w `onCreate()`, zamiast używać `onCreate()` do jego inicjalizacji i `onStart()` aby go uruchomić.
13. Skompiluj i uruchom aplikację. Zauważ, że stoper zaczyna działać, jak można się spodziewać.
14. Kliknij Home , aby zatrzymać aplikację. Minutnik przestaje działać, jak można się spodziewać.
15. Użyj recents screen , aby powrócić do aplikacji. Zauważ, że licznik czasu nie uruchamia się ponownie w tym przypadku, ponieważ `onCreate()` jest wywoływany tylko podczas uruchamiania aplikacji - nie jest wywoływany, gdy aplikacja powraca na pierwszy plan.

Najważniejsze kwestie do zapamiętania:

- Jeśli konfigurujesz jakieś zasoby w wywołaniu zwrotnym cyklu życia również obsłuż ten fakt przy zwalnianiu zasobów.
- Wykonaj konfigurację i porzucenie odpowiednimi metodami.
- Jeśli skonfigurujesz coś w `onStart()`,zatrzymaj go lub ponownie zburz w `onStop()`.



4. Zadanie: skorzystaj z biblioteki cyklu życia Androida (Android lifecycle library)

W aplikacji DessertClicker dość łatwo jest zauważyć, że jeśli uruchomiłeś stoper w `onStart()`,musisz zatrzymać stoper w `onStop()`.Jest tylko jeden timer, więc zatrzymanie timera nie jest trudne do zapamiętania.

W bardziej złożonej aplikacji na Androida możesz skonfigurować wiele rzeczy w `onStart()` lub `onCreate()`,a następnie rozerwać je wszystkie w `onStop()` lub `onDestroy()`.Na przykład możesz mieć animacje, muzykę, czujniki lub timery, których potrzebujesz zarówno do skonfigurowania, jak i zniszczenia, a także do uruchomienia i zatrzymania. Pominięcie jednego z nich prowadzi do błędów.

Biblioteka *lifecycle library,* która jest częścią Androida Jetpack, upraszcza to zadanie., simplifies this task. Biblioteka jest szczególnie przydatna w przypadkach, gdy trzeba śledzić wiele ruchomych części, z których niektóre znajdują się w różnych stanach cyklu życia. Biblioteka zmienia sposób działania cykli życia: zwykle działanie lub fragment mówi komponentowi (np. DessertTimer), co zrobić, gdy wystąpi wywołanie zwrotne cyklu życia. Ale kiedy korzystasz z biblioteki cyklu życia, sam komponent obserwuje zmiany cyklu życia, a następnie robi to, co jest potrzebne, gdy te zmiany się pojawią.

Istnieją trzy główne części biblioteki cyklu życia lifecycle library:

- Lifecycle *owners,* Właściciele cyklu życia, czyli elementy, które mają (a zatem „posiadają") cykl życia. `Activity` i `Fragment` są właścicielami cyklu życia. Właściciele Lifecycle implementują interfejs `LifecycleOwner`.
- Klasa `Lifecycle` , która utrzymuje aktualny stan właściciela cyklu życia i wyzwala zdarzenia, gdy nastąpią zmiany cyklu życia.
- Lifecycle *observers,* Obserwatorzy cyklu życia, którzy obserwują stan cyklu życia i wykonują zadania, gdy zmienia się cykl życia. Obserwatorzy cyklu życia implementują interfejs `LifecycleObserver`.

W tym zadaniu przekonwertuj aplikację DessertClicker, aby korzystała z biblioteki cyklu życia Android. Sprawdź jak biblioteka ułatwia zarządzanie cyklem życia aktywności i fragmentów.

Krok 1: Zmień DessertTimer w LifecycleObserver

Kluczową częścią biblioteki cyklu życia jest koncepcja obserwacji cyklu życia. Obserwacja pozwala klasom (takim jak DessertTimer) wiedzieć o cyklu życia aktywności lub fragmentu oraz uruchamiać się i zatrzymywać w odpowiedzi na zmiany tych stanów cyklu życia.

Za pomocą obserwatora cyklu życia możesz usunąć odpowiedzialność za uruchamianie i zatrzymywanie obiektów z metod aktywności i fragmentów.

1. Otwórz klasę `DesertTimer.kt`.
2. Zmień podpis klasy `DessertTimer` , aby wyglądał następująco:

```
class DessertTimer(lifecycle: Lifecycle) : LifecycleObserver {
```

Ta nowa definicja klasy ma dwie rzeczy:

- Konstruktor pobiera obiekt cyklu życia (`Lifecycle`), który jest cyklem życia obserwowanym przez timer.
- Definicja klasy implementuje interfejs `LifecycleObserver`.

3. Pod zmienną uruchamialną `runnable` dodaj blok `init` do definicji klasy. W bloku `init` block, użyj metody `addObserver()`, aby połączyć obiekt cyklu życia przekazany od właściciela (activity) do tej klasy (observer).

```
 init {
   lifecycle.addObserver(this)
}
```

4. Opisz `startTimer()` adnotacją `@OnLifecycleEvent annotation`, i użyj zdarzenia cyklu życia `ON_START` . Wszystkie zdarzenia cyklu życia, które może obserwować obserwator cyklu życia, należą do klasy `Lifecycle.Event`.

```
@OnLifecycleEvent(Lifecycle.Event.ON_START)
fun startTimer() {
```

5. Zrób to samo dla `stopTimer()`,używając zdarzenia `ON_STOP` event:

```
@OnLifecycleEvent(Lifecycle.Event.ON_STOP)
fun stopTimer()
```

Step 2: Zmodyfikuj MainActivity

Twoja klasa `MainActivity` jest już właścicielem cyklu życia poprzez dziedziczenie, ponieważ nadklasa `FragmentActivity` (superclass) implementuje `LifecycleOwner`. Dlatego nie musisz nic robić, aby Twoja aktywność była świadoma cyklu życia. Wszystko, co musisz zrobić, to przekazać obiekt cyklu życia działania do konstruktora `DessertTimer`.

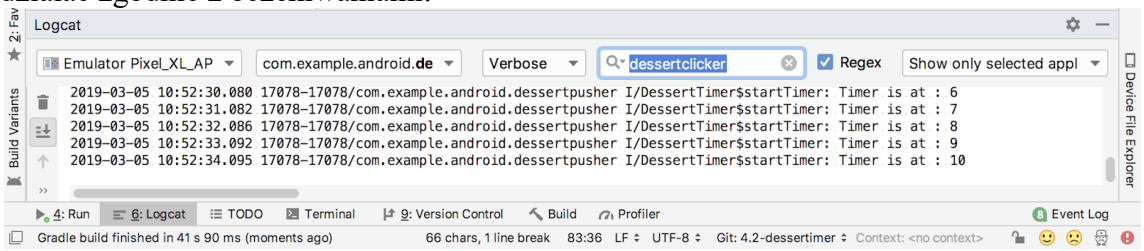
1. Otwórz `MainActivity`. W metodzie `onCreate()` zmodyfikuj inicjalizację `DessertTimer` , aby uwzględnić `this.lifecycle`:

```
dessertTimer = DessertTimer(this.lifecycle)
```

Właściwość activity `lifecycle` zawiera obiekt `Lifecycle` , do którego należy to activity.

2. Usuń wywołanie `startTimer()` w `onCreate()`,oraz wywołanie `stopTimer()` w `onStop()`.Nie musisz już mówić DessertTimerowi, co robić w działaniu, ponieważ DessertTimer obserwuje teraz sam cykl życia i jest automatycznie powiadamiany o zmianie stanu cyklu życia. Wszystko, co teraz robisz w tych wywołaniach zwrotnych, to rejestrowanie wiadomości.

3. Skompiluj i uruchom aplikację, a następnie otwórz Logcat. Zauważ, że stoper zaczął działać zgodnie z oczekiwaniami.



4. Kliknij przycisk strony głównej, aby umieścić aplikację w tle. Zauważ, że stoper przestał działać zgodnie z oczekiwaniami.

5. Zadanie: Symuluj zamknięcie aplikacji i użyj onSaveInstanceState()

Co stanie się z Twoją aplikacją i jej danymi, jeśli Android wyłączy tę aplikację, gdy będzie ona działała w tle? Ten trudny przypadek jest ważny do zrozumienia.

Gdy aplikacja przechodzi w tło, nie jest niszczona, jest tylko zatrzymywana i czeka na powrót użytkownika. Ale jednym z głównych problemów systemu operacyjnego Android jest utrzymanie płynności działania na pierwszym planie. Na przykład, jeśli użytkownik korzysta z aplikacji GPS, aby pomóc mu złapać autobus, ważne jest, aby szybko wyrenderować tę aplikację i nadal wyświetlać wskazówki. Mniej ważne jest, aby aplikacja DessertClicker, na którą użytkownik nie patrzył przez kilka dni, działała płynnie w tle.

Android reguluje aplikacje działające w tle, dzięki czemu aplikacja na pierwszym planie może działać bez problemów. Na przykład system Android ogranicza ilość przetwarzania, które mogą wykonywać aplikacje działające w tle.

Czasami Android nawet zamyka cały proces aplikacji, który obejmuje każdą aktywność związaną z aplikacją. Android dokonuje tego rodzaju zamykania, gdy system jest obciążony i grozi mu opóźnienie wizualne, więc w tym momencie nie są uruchamiane żadne dodatkowe połączenia zwrotne ani kod. Proces aplikacji jest po prostu cicho zamykany w tle. Ale dla użytkownika nie wygląda na to, że aplikacja została zamknięta. Gdy użytkownik powróci do aplikacji, którą system operacyjny zamknął, Android ponownie ją uruchomi.

W tym zadaniu symulujesz zamknięcie procesu Androida i sprawdzasz, co stanie się z Twoją aplikacją po ponownym uruchomieniu.

Note: Uwaga: przed rozpoczęciem upewnij się, że korzystasz z emulatora lub urządzenia obsługującego interfejs API 28 lub nowszy.

Krok 1: Użyj adb, aby zasymulować zamknięcie procesu

to narzędzie wiersza polecenia, które umożliwia wysyłanie instrukcji do emulatorów i urządzeń podłączonych do komputera. W tym kroku używasz adb, aby zamknąć proces aplikacji i zobaczyć, co się stanie, gdy Android wyłączy aplikację.

1. Skompiluj i uruchom aplikację. Kliknij kilka razy cupcake.

2. Naciśnij Home button , aby umieścić aplikację w tle. Twoja aplikacja jest teraz zatrzymana, a aplikacja może zostać zamknięta, jeśli Android potrzebuje zasobów, z których korzysta aplikacja.
3. W Android Studio kliknij kartę Terminal, aby otworzyć terminal wiersza polecenia.

4. Wpisz `adb` i naciśnij klawisz Return.

   Jeśli widzisz dużo danych wyjściowych, które zaczynają się od `Android Debug Bridge version X.XX.X` i kończą się tagami używanymi przez logcat (patrz logcat —help), wszystko jest w porządku. Jeśli zamiast tego zobaczysz, że polecenie `adb: command not found`, , upewnij się, że polecenie adb jest dostępne na ścieżce wykonania. Aby uzyskać instrukcje, zobacz „Dodawanie adb do ścieżki wykonania” w rozdziale Utilities chapter.
5. Skopiuj i wklej ten komentarz do wiersza poleceń i naciśnij klawisz Return:

```
adb shell am kill com.example.android.dessertclicker
```

To polecenie informuje wszystkie podłączone urządzenia lub emulatory, aby zatrzymały proces z nazwą pakietu deserclicker, ale tylko wtedy, gdy aplikacja jest w tle. Ponieważ aplikacja była w tle, na ekranie urządzenia lub emulatora nic się nie wyświetla, co oznacza, że proces został zatrzymany. W Android Studio kliknij kartę Uruchom, aby zobaczyć komunikat „Aplikacja zakończona”. Kliknij kartę Logcat, aby zobaczyć, że wywołanie zwrotne onDestroy () nigdy nie zostało uruchomione - twoja aktywność po prostu się zakończyła.

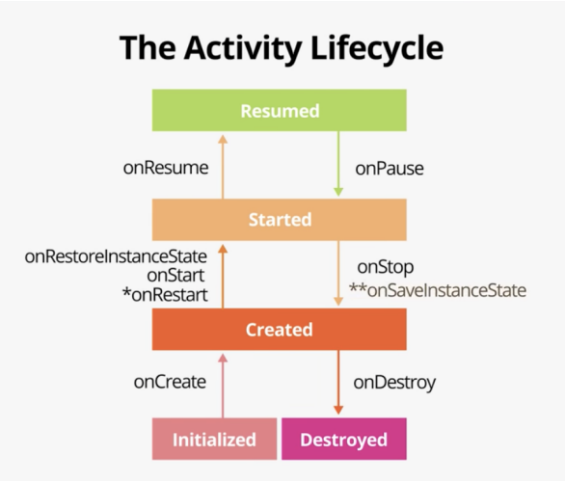
6. Użyj recents screen , aby powrócić do aplikacji. Twoja aplikacja pojawia się ostatnio, niezależnie od tego, czy została umieszczona w tle, czy też została całkowicie zatrzymana. Gdy wrócisz do aplikacji przy użyciu ekranu ostatnich, aktywność zostanie ponownie uruchomiona. Działanie przechodzi przez cały zestaw wywołań zwrotnych cyklu uruchamiania, w tym `onCreate()`.
7. Zauważ, że po ponownym uruchomieniu aplikacja resetuje twój „wynik” (zarówno liczbę sprzedanych deserów, jak i sumę dolarów) do wartości domyślnych (0). Jeśli Android zamknął twoją aplikację, dlaczego nie zapisał twojego stanu?

   Po ponownym uruchomieniu aplikacji przez system operacyjny Android stara się zresetować aplikację do stanu, w jakim była wcześniej. Android przyjmuje stan niektórych twoich widoków i zapisuje go w pakiecie, ilekroć odejdziesz od aktywności. Niektóre przykłady danych, które są automatycznie zapisywane, to tekst w EditText (o ile mają one ustawiony identyfikator w układzie) oraz tylny stos twojej aktywności.

   Czasami jednak system operacyjny Android nie wie o wszystkich danych. Na przykład, jeśli masz niestandardową zmienną, taką jak przychód w aplikacji DessertClicker, system operacyjny Android nie wie o tych danych ani ich znaczeniu dla Twojej aktywności. Musisz samodzielnie dodać te dane do pakietu.

Krok 2: Użyj onSaveInstanceState (), aby zapisać dane pakietu (bundle data)

Metoda `onSaveInstanceState()` to wywołanie zwrotne, którego używasz do zapisywania wszelkich danych, które mogą być potrzebne, jeśli system operacyjny Android zniszczy Twoją aplikację. Na diagramie wywołania zwrotnego cyklu życia funkcja `onSaveInstanceState()` jest wywoływana po zatrzymaniu działania. Jest wywoływany za każdym razem, gdy aplikacja przechodzi w tło.



Pomyśl o wywołaniu `onSaveInstanceState()` jako o środku bezpieczeństwa; daje to szansę na zapisanie niewielkiej ilości informacji w pakiecie, gdy Twoja aktywność wychodzi z pierwszego planu. System zapisuje teraz te dane, ponieważ jeśli czekał by do momentu zamknięcia aplikacji, mógłby być akurat pod presją zasobów. Zapisywanie danych za każdym razem zapewnia dostępność danych aktualizacji w pakiecie do przywrócenia, jeśli bedą potrzebne.

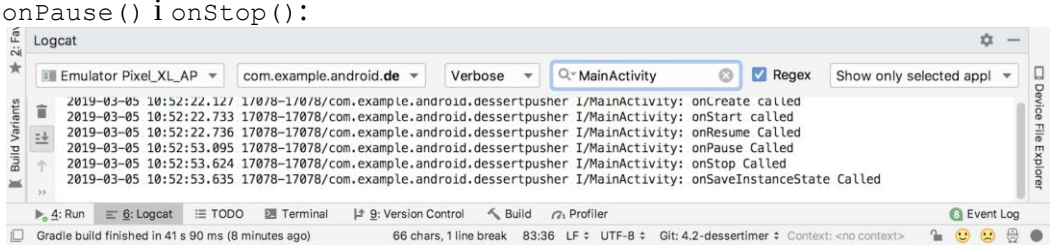
1. W `MainActivity`, przesłoń metodę `onSaveInstanceState()` i dodać instrukcję dziennika `Timber`.

```
override fun onSaveInstanceState(outState: Bundle) {
   super.onSaveInstanceState(outState)

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

Note: Uwaga: Istnieją dwa przesłonięcia dla `onSaveInstanceState()`,jedno tylko z parametrem `outState` i jedno zawierające parametry `outState` i `outPersistentState` parameters. Użyj tego pokazanego w powyższym kodzie, z pojedynczym parametrem `outState`.

2. Skompiluj i uruchom aplikację, a następnie kliknij przycisk **Home** , aby umieścić ją w tle. Zauważ, że wywołanie zwrotne `onSaveInstanceState()` występuje zaraz po

`onPause()` i `onStop()`:



3. W górnej części pliku, tuż przed definicją klasy, dodaj te stałe:

```
const val KEY_REVENUE = "revenue_key"
const val KEY_DESSERT_SOLD = "dessert_sold_key"
const val KEY_TIMER_SECONDS = "timer_seconds_key"
```

Użyjesz tych kluczy do zapisywania i pobierania danych z pakietu stanu instancji.

4. Przewiń w dół do `onSaveInstanceState()`,i zwróć uwagę na parametr `outState` który jest typu `Bundle`.

   Pakiet to zbiór par klucz-wartość, w którym klucze są zawsze ciągami znaków. W pakiecie można umieszczać prymitywne wartości, takie jak `int` i `boolean`

   Ponieważ system utrzymuje ten pakiet w pamięci RAM, najlepszą praktyką jest utrzymywanie małych danych w pakiecie. Rozmiar tego pakietu jest również ograniczony, choć rozmiar różni się w zależności od urządzenia. Zasadniczo powinieneś przechowywać znacznie mniej niż 100 tys., W przeciwnym razie ryzykujesz awarię aplikacji z błędem `TransactionTooLargeException`.
5. W funkcji `onSaveInstanceState()`,umieść wartość `revenue` (integer) w pakiecie za pomocą metody `putInt()`:

```
outState.putInt(KEY_REVENUE, revenue)
```

Metoda `putInt()` method (i podobne metody z klasy `Bundle` , takie jak `putFloat()` i `putString()` przyjmują dwa argumenty: ciąg znaków dla klucza (stała `KEY_REVENUE`) oraz rzeczywistą wartość do zapisania..

6. Powtórz ten sam proces z liczbą sprzedanych deserów i statusem timera:

```
outState.putInt(KEY_DESSERT_SOLD, dessertsSold)
outState.putInt(KEY_TIMER_SECONDS, dessertTimer.secondsCount)
```

Krok 3: Użyj onCreate(), aby przywrócić dane pakietu

1. Przewiń w górę do `onCreate()`,i sprawdź argumenty metody:

```
override fun onCreate(savedInstanceState: Bundle) {
```

Zauważ, że `onCreate()` otrzymuje `Bundle` przy każdym wywołaniu. Gdy aktywność zostanie ponownie uruchomiona z powodu zamknięcia procesu, zapisany pakiet jest przekazywany do funkcji `onCreate()`.Jeśli Twoja aktywność zaczynała się od nowa, ten pakiet `onCreate()` ma wartość NULL. Więc jeśli pakiet nie ma wartości NULL, wiesz, że „odtwarzasz” aktywność z wcześniej znanego punktu.

Note: Uwaga: Jeśli activity jest ponownie tworzona, wywołanie zwrotne `onRestoreInstanceState()` jest wywoływane po `on start()` , również z pakietem. W większości przypadków przywracasz stan aktywności w `onCreate()`.Ale ponieważ `onRestoreInstanceState()` jest wywoływany po `onStart()`, jeśli kiedykolwiek będziesz musiał przywrócić jakiś stan po wywołaniu `onCreate()` możesz użyć `onRestoreInstanceState()`.

2. Dodaj ten kod do `onCreate()`, po ustawieniu `DessertTimer`:

```
if (savedInstanceState != null) {
   revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
}
```

Test na wartość NULL określa, czy w pakiecie znajdują się dane lub czy pakiet ma wartość NULL, co z kolei informuje, czy aplikacja została uruchomiona na nowo, czy została utworzona ponownie po zamknięciu. Ten test jest częstym wzorcem przywracania danych z pakietu.
Zauważ, że klucz, którego użyłeś tutaj (`KEY_REVENUE`) jest tym samym kluczem, którego użyłeś do `putInt()`.Aby upewnić się, że używasz tego samego klucza za każdym razem, najlepszym rozwiązaniem jest zdefiniowanie tych kluczy jako stałych. Używasz `getInt()` aby pobrać dane z pakietu, tak samo jak `putInt()` aby umieścić dane w pakiecie. Metoda `getInt()` przyjmuje dwa argumenty:

- Ciąg znaków, który działa jak klucz, na przykład `"key_revenue"` dla wartości przychodu.
- Wartość domyślna w przypadku, gdy dla tego klucza w pakiecie nie ma żadnej wartości.

Liczba całkowita uzyskana z pakietu jest następnie przypisywana do zmiennej przychodu, a interfejs użytkownika użyje tej wartości.

3. Dodaj metody `getInt()`, aby przywrócić liczbę sprzedanych deserów i wartość timera:

```
if (savedInstanceState != null) {
   revenue = savedInstanceState.getInt(KEY_REVENUE, 0)dessertsSold =
savedInstanceState.getInt(KEY_DESSERT_SOLD, 0)
   dessertTimer.secondsCount =
       savedInstanceState.getInt(KEY_TIMER_SECONDS, 0)
}
```

4. Skompiluj i uruchom aplikację. Naciśnij babeczkę co najmniej pięć razy, aż zmieni się w pączek. Kliknij Home , aby umieścić aplikację w tle.
5. W Android Studio **Terminal** uruchom `adb` aby zamknąć proces aplikacji.

```
adb shell am kill com.example.android.dessertclicker
```

6. Użyj recents screen , aby powrócić do aplikacji. Zauważ, że tym razem aplikacja powraca z prawidłowymi przychodami i sprzedaje wartości sprzedane z pakietu. Ale zauważ też, że deser wrócił do babeczki. Pozostała jeszcze jedna rzecz, aby upewnić się, że aplikacja powróci z wyłączenia dokładnie tak, jak została wyłączona.

7. W `MainActivity`, sprawdź metodę `showCurrentDessert()` Zauważ, że ta metoda określa, który obraz deseru powinien zostać wyświetlony w działaniu na podstawie bieżącej liczby sprzedanych deserów i listy deserów w zmiennej `allDesserts`.

```
for (dessert in allDesserts) {
   if (dessertsSold >= dessert.startProductionAmount) {
       newDessert = dessert
   }
    else break
}
```

Wynik działania metody zależy od liczby sprzedanych deserów. Dlatego nie trzeba nic robić, aby przechowywać odwołanie do obrazu w pakiecie w funkcji `onSaveInstanceState` `W tym pakiecie przechowujesz już liczbę sprzedanych deserów`.

8. W `onCreate()`,w bloku, który przywraca stan z pakietu, wywołaj `showCurrentDessert()`:

```
 if (savedInstanceState != null) {
   revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
   dessertsSold = savedInstanceState.getInt(KEY_DESSERT_SOLD, 0)
   dessertTimer.secondsCount =
      savedInstanceState.getInt(KEY_TIMER_SECONDS, 0)
   showCurrentDessert()
}
```

9. Skompiluj i uruchom aplikację i umieść ją w tle. Użyj `adb` aby zamknąć proces. Użyj recents screen aby powrócić do aplikacji. Zauważ teraz, że zarówno podane wartości deserów, całkowity przychód, jak i obraz deserowy zostały poprawnie przywrócone.

6. Zadanie: badanie zmian konfiguracji (configuration chan ges)

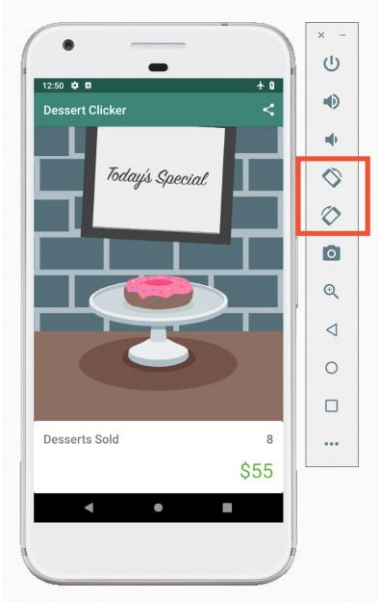
Jest jeszcze jeden szczególny przypadek w zarządzaniu cyklem życia aktywności i fragmentów, który należy zrozumieć: w jaki sposób zmiany konfiguracji wpływają na cykl życia twoich działań i fragmentów.

Zmiana konfiguracji ma miejsce, gdy stan urządzenia zmienia się tak radykalnie, że najłatwiejszym sposobem rozwiązania problemu przez system jest całkowite zamknięcie i odbudowanie activity. Na przykład, jeśli użytkownik zmieni język urządzenia, cały układ może wymagać zmiany, aby uwzględnić różne kierunki tekstu. Jeśli użytkownik podłączy urządzenie do stacji dokującej lub doda fizyczną klawiaturę, układ aplikacji może wymagać skorzystania z innego rozmiaru wyświetlacza lub układu. A jeśli orientacja urządzenia ulegnie zmianie - jeśli urządzenie zostanie obrócone z pionowej na poziomą lub odwrotnie - układ może wymagać zmiany w celu dopasowania do nowej orientacji.

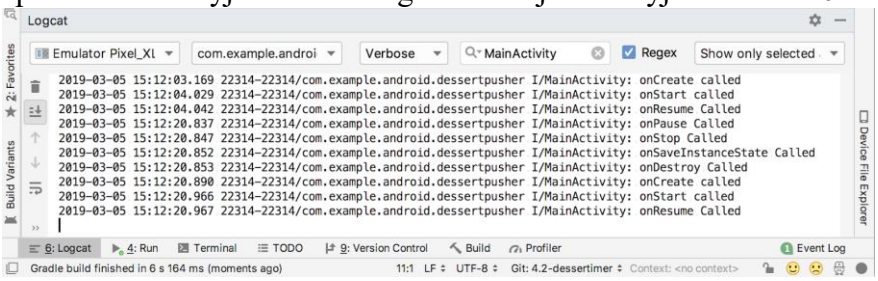
Krok 1: Zbadaj rotację urządzenia i wywołania zwrotne cyklu życia

1. Skompiluj i uruchom aplikację, a następnie otwórz Logcat.

2. Obróć urządzenie lub emulator do trybu poziomego. Emulator można obracać w lewo lub w prawo za pomocą przycisków obrotowych lub klawiszy Control i strzałek.



3. Sprawdź dane wyjściowe w Logcat. Filtruj dane wyjściowe na `MainActivity`.



Zauważ, że gdy urządzenie lub emulator obraca ekran, system wywołuje wszystkie wywołania zwrotne cyklu życia w celu zamknięcia działania. Następnie, po ponownym utworzeniu działania, system wywołuje wszystkie wywołania zwrotne cyklu życia, aby rozpocząć działanie.

4. W `MainActivity`, ZAkomentuj całą metodę `onSaveInstanceState()`.
5. Skompiluj i uruchom ponownie aplikację. Kliknij cupcake kilka razy i obróć urządzenie lub emulator. Tym razem, gdy urządzenie zostanie obrócone, a aktywność zostanie zamknięta i ponownie utworzona, aktywność rozpocznie się z wartościami domyślnymi.

   Gdy nastąpi zmiana konfiguracji, system Android używa tego samego pakietu stanu wystąpienia, o którym dowiedziałeś się w poprzednim zadaniu, aby zapisać i przywrócić stan aplikacji. Podobnie jak w przypadku zamykania procesu, użyj `onSaveInstanceState()`, aby umieścić dane aplikacji w pakiecie. Następnie przywróć dane w funkcji `onCreate()`, aby uniknąć utraty danych stanu aktywności, jeśli urządzenie zostanie obrócone.
6. W `MainActivity`, usuń komentarz z metody `onSaveInstanceState()` uruchom aplikację, kliknij cupcake i obróć aplikację lub urządzenie. Zauważ, że tym razem dane deserowe są zachowywane podczas rotacji aktywności.