Service, BroadcastReceiver

Service(serwis) jest komponentem androida, który służy do wykonywania działań w tle. Najczęściej są to czynności wymagające dłuższego działania – synchronizacja danych, pobieranie plików, odtwarzanie muzyki.

Rodzaje serwisów:

Started Service (Uruchamiany):

Rozpoczynany za pomocą startService().

Działa, dopóki nie zostanie zatrzymany ręcznie przez stopSelf() lub stopService().

Bound Service (Powiązany):

Powiązany z komponentem (np. aktywnością) za pomocą bindService().

Działa tak długo, jak jest powiązany z innym komponentem.

BroadcastReceiver(odbiornik) jest komponentem, który nasłuchuje wiadomości (broadcast) wysyłane przez system Android lub inne aplikacje. Wykonuje zadania w odpowiedzi na odebraną wiadomość. Wiadomości systemowe mogą dotyczyć np dostępu do sieci, czy niskiego stanu baterii. Aplikacja może powiadamiać np. o zakończeniu pobierania pliku.

Serwis musi zostać zarejestrowany w pliku manifestu:

```
<service
    android:name=".MyService"
    android:enabled="true"
    android:exported="false" />
```

`android:enabled` -serwis aktywowany
`android:exported` -serwis dostępny tylko dla aplikacji

BroadcastReceiver można zarejestrować w pliku manifestu, jak również dynamicznie w kodzie. W przykładzie BroadcastReceiver będzie rejestrowany dynamicznie, a w manifeście rejestracja mogła wyglądać następująco:

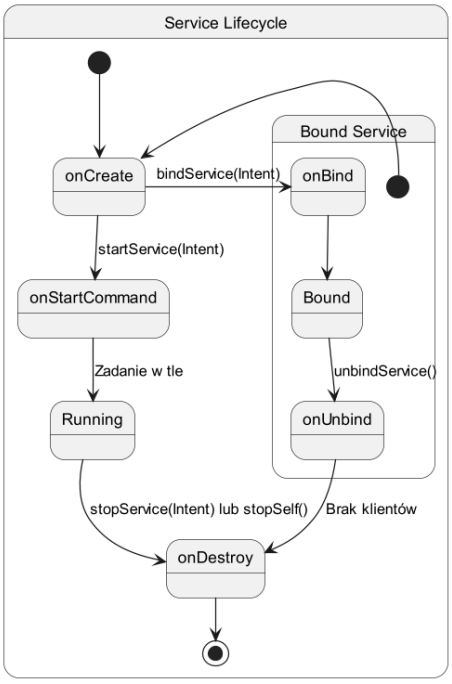
```
<receiver
    android:name=".MyBroadcastReceiver"
    android:enabled="true"
    android:exported="false">
    <intent-filter>
        <action        android:name="com.example.DATA_DOWNLOADED"        />
    </intent-filter>
</receiver>
```

`<action android:name="com.example.DATA_DOWNLOADED" />`- akcja jaką nasłuchuje BroadcastReceiver.
`android:exported="false"`- działanie tylko w obrębie aplikacji.

BroadcastReceiver należy rejestrować statycznie jeśli ma reagować nawet wtedy, gdy aplikacja jest zamknięta lub w tle. **Uwaga** od Androida 8 większość niejawnych Broadcastów **nie będzie** obsługiwana za pomocą rejestracji odbiornika w manifeście.
Tutaj wyjątki:
https://developer.android.com/develop/background-work/background-tasks/broadcasts/broadcast-exceptions

	Statyczna rejestracja (Manifest)	**Dynamiczna rejestracja (np. DisposableEffect)**
Zasięg działania	Reaguje zawsze, nawet gdy aplikacja jest zamknięta.	Działa tylko, gdy aplikacja (lub jej komponent) jest aktywny.
Broadcast systemowy	Zawsze obsługuje zdarzenia globalne(**UWAGA na android 8**)	Wymaga, aby aplikacja była uruchomiona.
Kontrola cyklu życia	Brak – działa zawsze, gdy broadcast jest wysłany.	Pełna kontrola – działa tylko w aktywnej części aplikacji.

Cykl życia serwisu:



Działanie serwisu w trybie "Started Service" (Uruchamiany):

startService(Intent):
Wywołuje onCreate (tylko raz, jeśli serwis jeszcze nie istnieje).
Wywołuje onStartCommand (za każdym razem, gdy serwis jest uruchamiany).
Serwis wykonuje zadanie w tle.
stopSelf() lub stopService(Intent):
Wywołuje onDestroy.

Działanie serwisu w trybie "Bound Service" (Powiązany):

bindService(Intent, ServiceConnection, int):

Wywołuje onCreate (jeśli serwis jeszcze nie istnieje).

Wywołuje onBind.

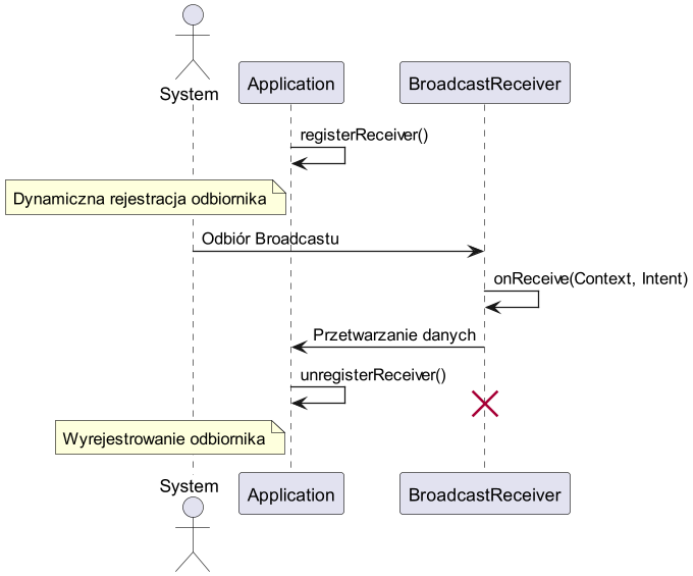
Klient korzysta z serwisu (np. wywołuje metody przez IBinder).

unbindService(ServiceConnection):

Wywołuje onUnbind.

Jeśli nie ma innych klientów, serwis wywołuje onDestroy.

Cykl życia BroadcastReceiver:



Rejestracja odbiornika:

Aplikacja dynamicznie rejestruje BroadcastReceiver za pomocą metody registerReceiver.

Wysłanie broadcastu:

Android wysyła broadcast, który pasuje do IntentFilter ustawionego podczas rejestracji odbiornika.

Odbiór broadcastu:

BroadcastReceiver odbiera broadcast i uruchamia swoją metodę onReceive.W metodzie onReceive przetwarzane są dane z broadcastu (np. z Intent), a wyniki są przesyłane do aplikacji (np. do UI).

Wyrejestrowanie odbiornika:

Po zakończeniu użytkowania, aplikacja wyrejestrowuje odbiornik za pomocą metody unregisterReceiver.

Połączenie BroadcastReceiver oraz Serivce:

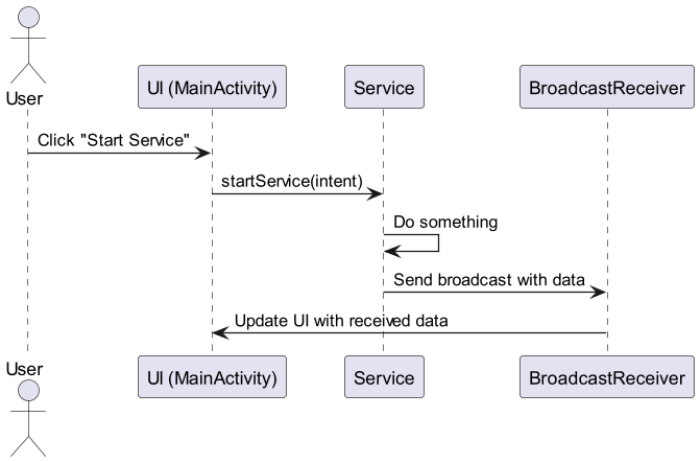
Te dwa komponenty często pracują razem.
BroadcastReceiver i Service mogą współpracować w obie strony:

-Serwis może wysyłać broadcasty – na przykład informując o stanie realizowanego zadania.

-BroadcastReceiver może uruchamiać serwis – na przykład w reakcji na zdarzenie systemowe lub aplikacyjne.

Przykład kierunku Service -> BroadcastReceiver:

Użytkownik klika przycisk „Start Service, serwis pobiera dane, a po zakończeniu wysyła broadcast. Odbiornik(BroadcastReceiver) odbiera dane, UI zostaje odświeżone.



Przykład serwisu, zadaniem będzie symulowanie pobieranie plików:

```
import android.app.Service
import android.content.Intent
import android.os.Handler
import android.os.IBinder
import android.util.Log


/**
 * Serwis będzie symulował pobieranie plików
 */
class MyService : Service() {
    private val handler = Handler()
    private var currentFile = 0
    private val total = 5

    private val data = mutableListOf<String>()

    override fun onStartCommand(intent: Intent?, flags: Int, startId: Int):
Int{
        Log.d("MyService -------->", "started")
        download()
        return START_STICKY
    }
```

```
    private fun download() {
        if (currentFile<total){
            Log.d("MyService ----------->", "downloading file ...
$currentFile")
            handler.postDelayed( {download()},1000)
            data.add("File $currentFile")
            currentFile++
        }else{
            Log.d("MyService ----------->", "END")
            val broadcastIntent = Intent("com.example.DATA_DOWNLOADED")
            broadcastIntent.putStringArrayListExtra("DATA_LIST",
ArrayList(data))
            sendBroadcast(Intent(broadcastIntent))
            stopSelf()
        }

    }

    override fun onBind(p0: Intent?): IBinder? {
        Log.d("MyService ----------->", "onBind()")
        return null
    }

    override fun onDestroy() {
        super.onDestroy()
        handler.removeCallbacksAndMessages(null)
        Log.d("MyService ----------->", "onDestroy()")
    }
}
```

Metoda **onStartCommand** jest wywoływana, gdy serwis zostaje uruchomiony za pomocą startService(Intent).

Parametry:
intent: Intent?
Intencja (Intent), która została przesłana do serwisu podczas jego uruchomienia.Może zawierać dodatkowe dane, np. parametry określające zadanie, które serwis ma wykonać.
flags: Int
Flagi informujące o sposobie uruchomienia serwisu.
Przykładowe wartości:
START_FLAG_REDELIVERY: System ponownie dostarczył ten sam Intent, ponieważ serwis został wcześniej nieoczekiwanie przerwany.
START_FLAG_RETRY: System ponawia próbę uruchomienia serwisu
startId: Int
Unikalny identyfikator dla konkretnego wywołania serwisu.
return START_STICKY - Serwis zostanie automatycznie ponownie uruchomiony przez system po jego zatrzymaniu.
Android generuje flagi na podstawie aktualnej sytuacji.

Metoda **download()** symuluje pobranie 5 plików, używając Handlera – co sekundę wywołuje samą siebie dodając +1 do licznika plików, oraz dodaje nową wartość do listy.
Po symulacji tworzony jest obiekt Intent `broadcastIntent`, który zawiera akcję "com.example.DATA_DOWNLOADED". Do Intent dodawana jest lista „pobranych” plików jako ArrayList:

```
broadcastIntent.putStringArrayListExtra("DATA_LIST",
ArrayList(data))
```

Funkcja **sendBroadcast** wysyła broadcast do wszystkich zarejestrowanych odbiorników (BroadcastReceiver), które nasłuchują na akcję "com.example.DATA_DOWNLOADED".
Po zakończeniu zadania serwis zatrzymuje sam siebie, wywołując stopSelf.

`onBind` zwraca null, jest to serwis uruchamiany(started services) przez startService(Intent) i nie wymaga komunikacji z klientem po uruchomieniu.
`onDestroy` jest wywoływana przez system, gdy serwis jest niszczony, dodatkowo usuwa wszystkie zaplanowane zadania w Handler.

Prosty odbiornik, który dziedziczy po BroadcastReceiver:

```
import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.util.Log

class MyBroadcastReceiver(private val onDataRecived: (List<String>)->Unit)
: BroadcastReceiver() {
    override fun onReceive(p0: Context?, p1: Intent?) {
      if (p1?.action == "com.example.DATA_DOWNLOADED"){
          Log.d("MyBroadcastReceiver", "Data downloaded")
          val dataList = p1.getStringArrayListExtra("DATA_LIST")
          if (dataList != null) {
              onDataRecived(dataList)
          }
      }
    }
}
```

Należy zaimplementować metodę onReceive, dodatkowo odbiornik przyjmuje metodę, której parametrem będzie lista String i nie będzie zwracać żadnej wartości `onDataRecived` .

Jeżeli akcja z intencji będzie równa : `"com.example.DATA_DOWNLOADED")` . odbiornik pobierze listę i przekaże ją do funkcji `onDataRecived(dataList)` jeżeli lista nie będzie pusta .
Metoda zostanie przekazana jako parametr, co czyni odbiornik elastycznym – w dowolny sposób będzie można powiadomić UI o zmianach.

Rejestracja serwisu, oraz update UI:

```
import android.content.Context.RECEIVER_EXPORTED
import android.content.Intent
import android.content.IntentFilter
import android.os.Build
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.DisposableEffect
import androidx.compose.runtime.getValue
```

```
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp


@Composable
fun ReceiverUI(){
    //context composable:
    val context = LocalContext.current
    var someData by remember { mutableStateOf(emptyList<String>()) }
    //intent uruchomienia serwisu:
    val myServiceIntent = Intent(context,MyService::class.java)
//.setPackage(context.packageName)

    // Rejestracja odbiornika MyBroadcastReceiver
    DisposableEffect(Unit) {
        val receiver = MyBroadcastReceiver{ newData ->
            someData = newData // Aktualizacja danych: funkcja onReceived
        }
        val intentFilter = IntentFilter("com.example.DATA_DOWNLOADED")

        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            context.registerReceiver(receiver, intentFilter,
RECEIVER_EXPORTED)
        } else {
            @Suppress("UnspecifiedRegisterReceiverFlag")
            context.registerReceiver(receiver, intentFilter)
        }

        onDispose {
            context.unregisterReceiver(receiver) // Usuwanie odbiornika po
zakończeniu
        }
    }

    //UI:
    Column(modifier = Modifier.fillMaxSize()){
        Button(
            onClick = {context.startService(myServiceIntent)}
        ) {
            Text("Start download service")
        }
        LazyColumn(modifier = Modifier.fillMaxSize()) {
            items(someData){
                item -> Text(item, modifier = Modifier.padding(16.dp))
            }
        }
    }
}
```

Uruchomienie serwisu:

Pamiętamy o kontekscie Composable:

```
val context = LocalContext.current
```

Tworzymy intencję uruchomienia serwisu:

```
val myServiceIntent = Intent(context,MyService::class.java)
```

Serwis będzie uruchamiany na kliknięcie przycisku:

```
onClick = {context.startService(myServiceIntent)}
```

Rejestracja MyBroadcastReceiver dynamicznie:

Używamy DisposableEffect - Wykona akcję podczas tworzenia kompozycji – zarejestruje odbiornik, oraz przy odmontowaniu kompozycji – wyrejestruje.

```
    DisposableEffect(Unit) {
//…
```

Wywołujemy konstruktor MyBroadcastReceiver przekazując funkcję onDataReceived za pomocą labmdy:

```
        val receiver = MyBroadcastReceiver{ newData ->
            someData = newData // Aktualizacja danych: funkcja onReceived
        }
```

Funkcja aktualizuje zmienną someData, która jest zarządzana przez jetpack compose:

```
var someData by remember { mutableStateOf(emptyList<String>()) }
```

Należy wyszukać odpowiedniej dodając do rejestracji odbiornika IntentFilter:

```
val intentFilter = IntentFilter("com.example.DATA_DOWNLOADED")
```

Uwaga!!!

W **Androidzie 13** wprowadzono obowiązek określenia jednej z flag:
RECEIVER_EXPORTED: Odbiornik jest eksportowany (może odbierać broadcasty spoza aplikacji).
RECEIVER_NOT_EXPORTED: Odbiornik nie jest eksportowany (może odbierać tylko broadcasty wewnątrz aplikacji).

Rejestracja będzie się różniła w zależności od wersji androida:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    context.registerReceiver(receiver, intentFilter, RECEIVER_EXPORTED)
} else {
    @Suppress("UnspecifiedRegisterReceiverFlag")
    context.registerReceiver(receiver, intentFilter)
}
```

@Suppress("UnspecifiedRegisterReceiverFlag") wyłącza ostrzeżenia kompilatora, brak flag na starszych wersjach jest poprawny.

Zadanie:

Napisz aplikację, w której serwis, który będzie symulował pobierał losowo 1 książkę (np. z 5 przygotowanych tytułów).
Książkę może reprezentować mapa z tytułem i jej opisem/fragmentem.
Przykładowe książki:
https://www.gutenberg.org/browse/scores/top
Aplikacja powinna zawierać również UI w postaci tabeli z danymi na temat pobranych książek.

Kiedy serwis skończy symulację pobieranie, wyśle za pomocą Broadcastu nazwę książki, oraz statystyki tych książek: liczba słów, liczba liter, najczęściej występujące słowo.

Dodatkowo serwis po pobraniu wyśle powiadomienie o pobraniu książki.

Odbiorca powinien sprawdzić czy w tabeli już jest taka książka, jeżeli tak to nie będzie wstawiał odebranych danych, w innym wypadku wstawi dane do listy i zaktualizuje tabelę