

Kotlin & Android w Android Studio (3.5.1)

Multimedia

Tym razem zajmiemy się multimediami w naszym telefonie. Utworzymy aplikację, która demonstrować będzie sposób obsługi aparatu fotograficznego, zajmiemy się też tworzeniem galerii wyświetlającej obrazy zawarte w naszym telefonie. Ponadto zbudujemy odtwarzacz plików dźwiękowych i filmów wideo.

Tworzenie aplikacji wykorzystującej multimedia

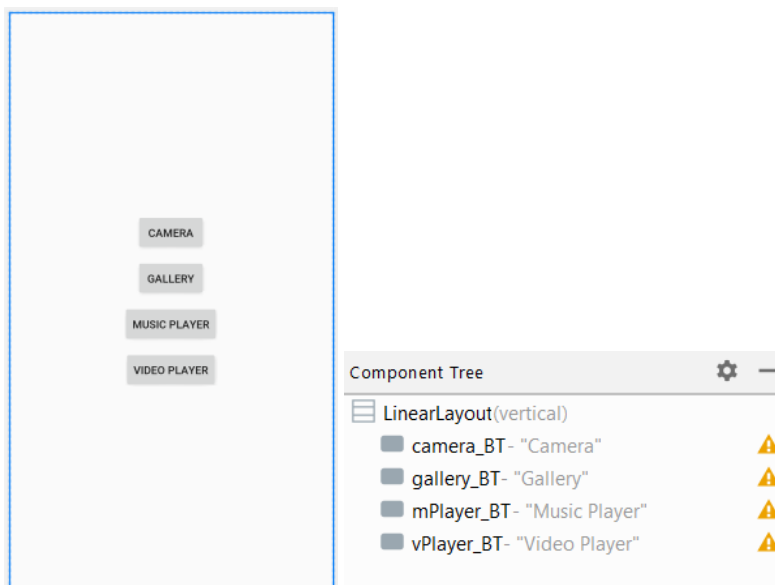
Nasza aplikacja bazować będzie na kilku aktywnościach i kilku layoutach. Zaczynamy od utworzenia głównego layoutu programu.

ZADANIE

Utwórz nowy projekt Android Studio o nazwie **MultimediaApp**, zawierający **EmptyActivity**. W pliku layoutu głównej aktywności użyj komponentu **LinearLayout(vertical)**. Ustaw dla layoutu parametr **gravity** na wartość **center**. Dodaj 4 przyciski **Button** zawierające następujące teksty:

- Camera (id = camera_BT);
- Gallery (id = gallery_BT);
- Music Player (id = mPlayer_BT);
- Video Player (id = vPlayer_BT).

Dla każdego przycisku ustaw: margin = 5dp, layout_width = wrap_content.



Przyciski posłużą nam do wywoływania aktywności odpowiedzialnych za realizację danych funkcjonalności naszej aplikacji.

Utwórzmy teraz pliki aktywności i layoutów dla każdej z funkcjonalności. Kod tych plików będziemy modyfikować nieco później.

ZADANIE

Utwórz w naszym projekcie 4 nowe pliki aktywności **EmptyActivity** o następujących nazwach:

- CameraActivity (z plikiem layoutu activity_camera);
- GalleryActivity (z plikiem layoutu activity_gallery);
- MusicActivity (z plikiem layoutu activity_music);
- VideoActivity (z plikiem layoutu activity_video).

Przechodzimy do oprogramowania przycisków głównego layoutu aby uruchamiały wyżej utworzone aktywności.

ZADANIE

Zaimplementuj kod obsługi przycisków głównej aktywności.

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    camera_BT.setOnClickListener { it: View!  
        val intentCamera = Intent(applicationContext, CameraActivity::class.java)  
        startActivity(intentCamera)  
    }  
  
    gallery_BT.setOnClickListener { it: View!  
        val intentGallery = Intent(applicationContext, GalleryActivity::class.java)  
        startActivity(intentGallery)  
    }  
  
    mPlayer_BT.setOnClickListener { it: View!  
        val intentMusic = Intent(applicationContext, MusicActivity::class.java)  
        startActivity(intentMusic)  
    }  
  
    vPlayer_BT.setOnClickListener { it: View!  
        val intentVideo = Intent(applicationContext, VideoActivity::class.java)  
        startActivity(intentVideo)  
    }  
}
```

Ponieważ w naszej aplikacji będziemy potrzebować dostępu do aparatu oraz do plików dodajmy w pliku manifestu odpowiednie uprawnienia.

ZADANIE

Dodaj w pliku **AndroidManifest** poniższy kod uprawnień naszej aplikacji:

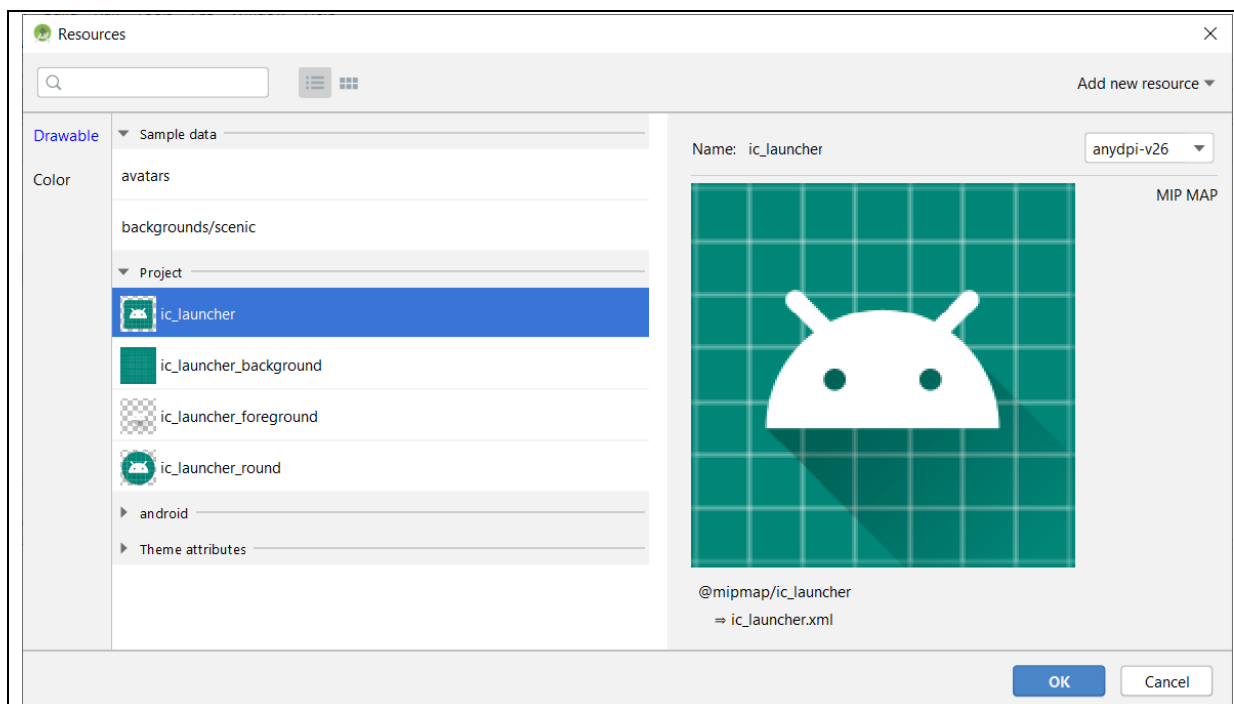
```
<uses-permission android:name="android.permission.CAMERA"/>  
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>  
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Obsługa aparatu

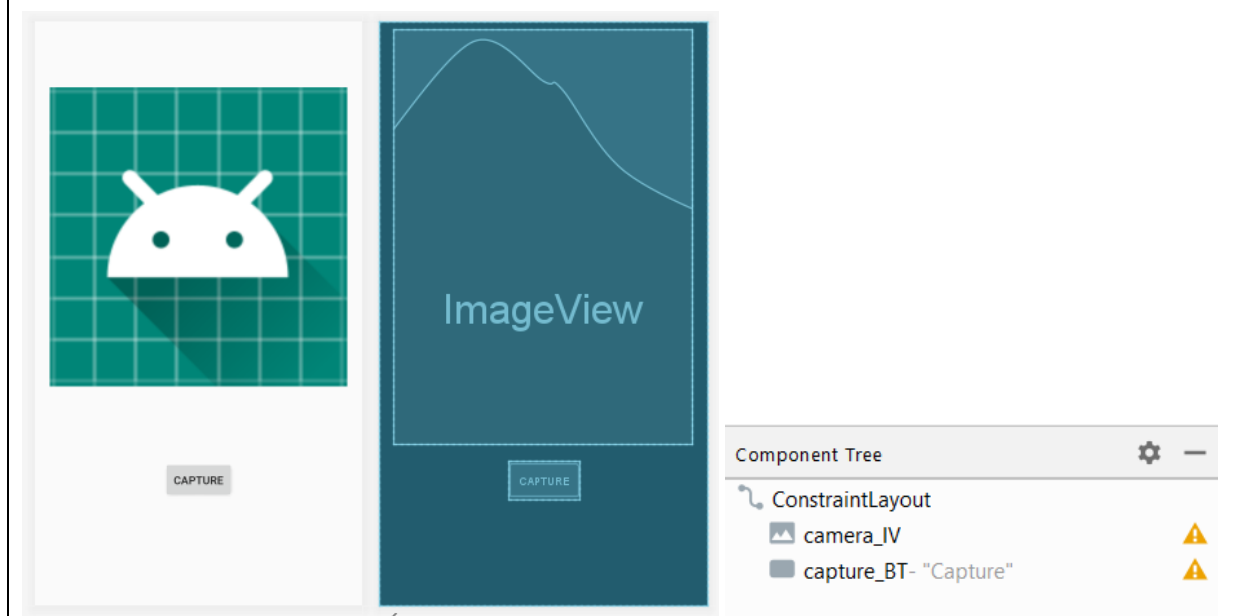
W tej sekcji zajmiemy się obsługą aparatu. Przygotujmy layout dla tej aktywności.

ZADANIE

Dodaj do layoutu **activity_camera** komponent **ImageView** i ustaw w nim obraz **ic_launcher**.



Dodaj również przycisk i ustaw na nim tekst **Capture**. Ustaw identyfikatory komponentów: dla **ImageView** id=camera_IV, dla **Button** id=capture_BT. Dopasuj rozmiar komponentów i ich położenie.



Layout aktywności obsługi aparatu jest gotowy. Teraz trzeba zaimplementować obsługę przycisku **Capture**. Skorzystamy tutaj z intencji wywołującej aplikację Aparat.

ZADANIE

Dodaj w **CameraActivity** kod odpowiedzialny za wykonywanie zdjęć.

```

class CameraActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_camera)

        capture_BT.setOnClickListener { it: View!
            // intencja wywołująca aplikację Aparat
            var intentCapture = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
            // uruchomienie intencji z możliwością dostępu do rezultatu
            // jej działania - w tym przypadku do wykonanego zdjęcia
            startActivityForResult(intentCapture, requestCode: 123)
        }

        // funkcja przetwarzająca wynik działania intencji
        override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
            super.onActivityResult(requestCode, resultCode, data)
            if(data!=null && requestCode==123){
                var bmp = data.extras.get("data") as Bitmap // pobranie obrazu
                camera_IV.setImageBitmap(bmp) // ustawienie obrazu w ImageView
            }
        }
    }
}

```

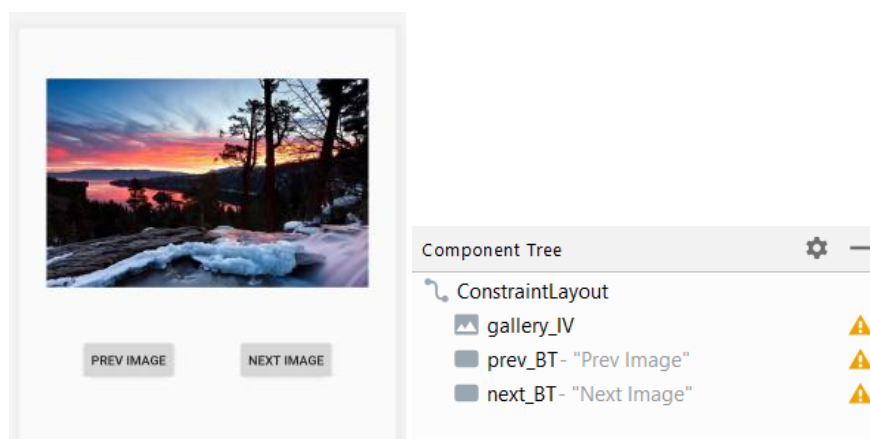
Korzystamy z metody **startActivityForResult()**, która nie tylko uruchamia intencję ale również pozwala na dostęp do wyniku jej działania. Do naszej intencji przypisujemy kod '123' aby później móc odnieść się do niej. Następnie w metodzie **onActivityResult()** sprawdzamy czy dane będące rezultatem działania intencji są niepuste (**data!=null**) oraz zgodność kodu nadanego wcześniej intencji (wynikowi jej działania). Ostatecznie przetwarzamy otrzymany rezultat – w naszym przypadku wyświetlamy zdjęcie w **ImageView**.

Tworzenie galerii zdjęć

Teraz zajmiemy się galerią zdjęć. Zaczynamy od zbudowania layoutu. Nasza galeria zbudowana będzie z komponentu **ImageView** do wyświetlania obrazów oraz 2 przycisków umożliwiających nawigowanie.

ZADANIE

Dodaj do layoutu **activity_gallery** komponent **ImageView** oraz 2 przyciski **Button** wg poniższego rysunku. Wprowadź identyfikatory komponentów i ustaw wyświetlany na nich tekst.



Mając gotowy layout uzupełniamy kod aktywności galerii. Na początku musimy zbudować listę wszystkich plików graficznych (ścieżek do plików) znajdujących się w pamięci telefonu aby następnie móc swobodnie nawigować pomiędzy nimi. W tym celu utworzymy funkcję, która pozwoli nam utworzyć odpowiednią strukturę.

ZADANIE

Zaimplementuj w **Gallery_Activity** poniższą funkcję.

```
fun getAllImages(): ArrayList<String> {  
    val fileList: ArrayList<String> = ArrayList()  
    // katalog główny urządzenia  
    var path: String = Environment.getExternalStorageDirectory().absolutePath  
    // petla przechodząca po wszystkich katalogach, podkatalogach i plikach  
    File(path).walk().forEach { it: File  
        if(it.toString().endsWith( suffix: ".jpg")) // filtr plików JPG  
            fileList.add(it.toString())  
    }  
    return fileList  
}
```

Powyższa funkcja tworzy ArrayListę ścieżek do plików graficznych w formacie JPG. Teraz wystarczy wywołać naszą funkcję a następnie oprogramować akcje przycisków nawigacyjnych naszej galerii.

ZADANIE

Zaimplementuj w metodzie **onCreate()** następujący kod:

```
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_gallery)  
  
    // ArrayLista wszystkich obrazów  
    val allImagesPaths = getAllImages()  
    // indeks aktualnie wyświetlanego obrazu  
    var currImgIndex = 0  
    // wyświetlenie pierwszego obrazu (indeks 0)  
    gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex)))  
  
    // przejście do wyświetlania poprzedniego obrazu  
    prev_BT.setOnClickListener { it: View!  
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex - 1)))  
        currImgIndex--  
    }  
  
    // przejście do wyświetlania kolejnego obrazu  
    next_BT.setOnClickListener { it: View!  
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex + 1)))  
        currImgIndex++  
    }  
}
```

Nasza aplikacja wyszukuje obrazy i wyświetla je oraz umożliwia nawigowanie pomiędzy nimi. Jednak trzeba dodać pewne zabezpieczenia:

- Nie możemy wyświetlać obrazów jeżeli nie ma ich w telefonie. Wówczas przyciski powinny być nieaktywne.
- Jeśli istnieje tylko 1 obraz to wyświetlamy go, ale przyciski pozostają nieaktywne.
- Przy wyświetlaniu pierwszego obrazu z listy przycisk **Prev Image** powinien pozostać nieaktywny.

- Przy wyświetlaniu ostatniego pliku z listy przycisk **Next Image** powinien być nieaktywny.
- Odblokowanie przycisków nawigacyjnych powinno nastąpić po opuszczeniu skrajnych pozycji listy – odpowiednio **Prev Image** gdy jesteśmy za indeksem 0 oraz **Next Image** będąc przed ostatnim indeksem.

ZADANIE

Zmodyfikuj kod metody **onCreate()** wg poniższych obrazów.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_gallery)

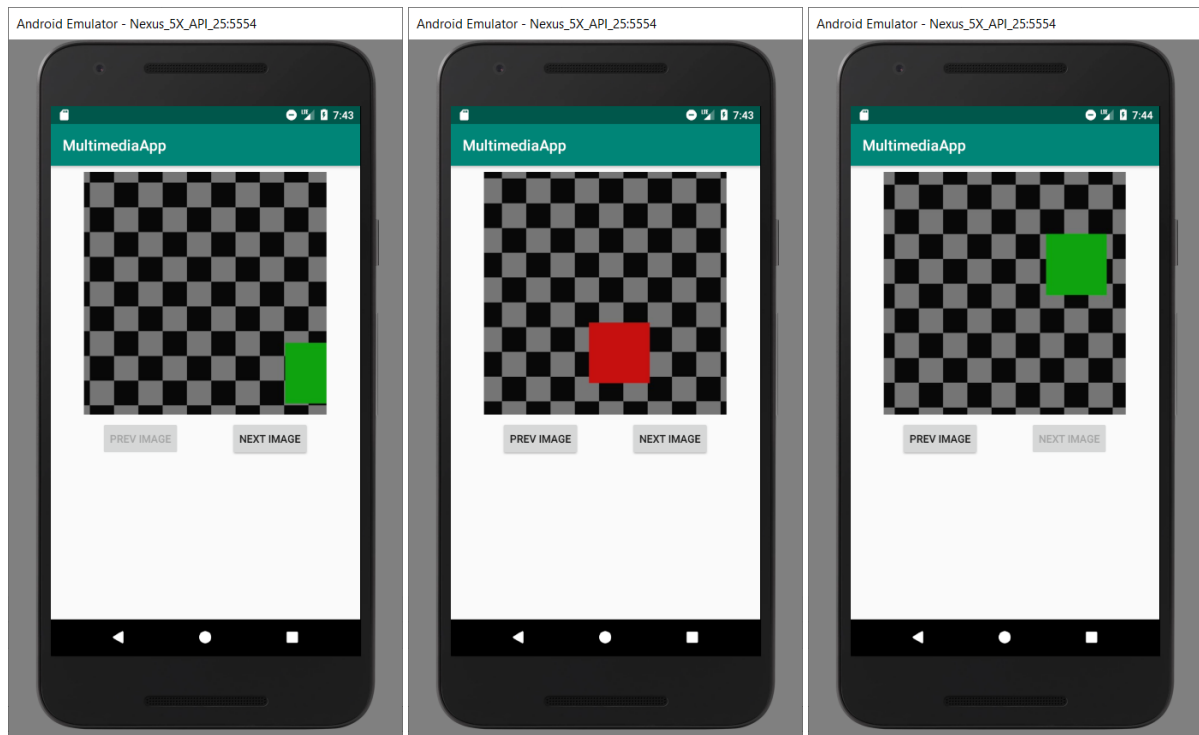
    // ArrayLista wszystkich obrazow
    val allImagesPaths = getAllImages()
    // indeks aktualnie wyswietlanego obrazu
    var currImgIndex = 0

    if(allImagesPaths.size>0)
        // wyswietlenie pierwszego obrazu (indeks 0)
        gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex)))
    prev_BT.isEnabled = false // na starcie przycisk nieaktywny
    if(allImagesPaths.size<=1)
        next_BT.isEnabled=false

    // przejście do wyswietlania poprzedniego obrazu
    prev_BT.setOnClickListener { it: View!
        if(currImgIndex>0) {
            gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex - 1)))
            currImgIndex--
            if(!next_BT.isEnabled)
                next_BT.isEnabled=true
        }
        if(currImgIndex==0)
            prev_BT.isEnabled=false
    }

    // przejście do wyswietlania kolejnego obrazu
    next_BT.setOnClickListener { it: View!
        if(currImgIndex<allImagesPaths.size-1) {
            gallery_IV.setImageURI(Uri.parse(allImagesPaths.get(currImgIndex + 1)))
            currImgIndex++
            if(!prev_BT.isEnabled)
                prev_BT.isEnabled=true
        }
        if(currImgIndex==allImagesPaths.size-1)
            next_BT.isEnabled=false
    }
}
```

Galeria gotowa, możemy bezpiecznie poruszać się między obrazami.

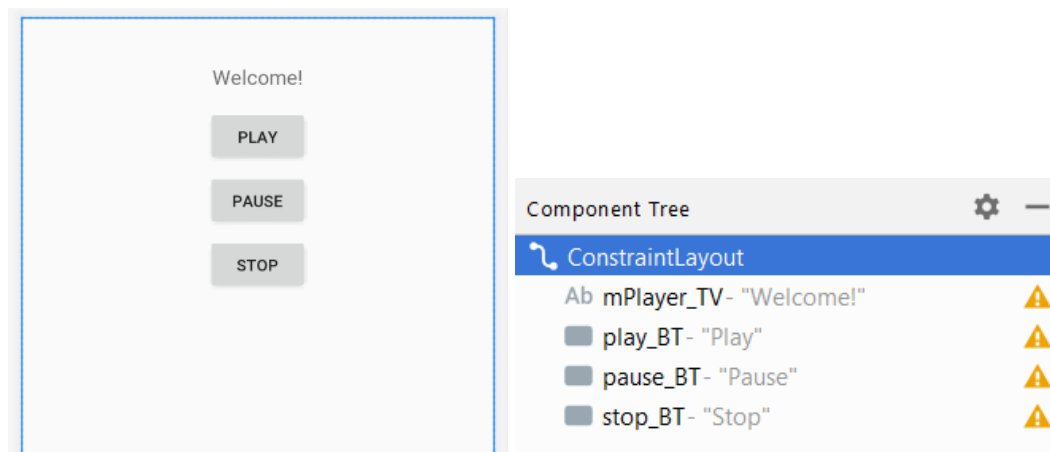


Odtwarzanie muzyki

W celu utworzenia prostego odtwarzacza muzyki skorzystamy z klasy **MediaPlayer**, która udostępnia gotowe metody do odtwarzania dźwięku. Zanim jednak zajmiemy się samym dźwiękiem przygotujmy layout naszego playera. Będzie on zbudowany z komponentu **TextView** do wyświetlania informacji o stanie playera oraz 3 przyciski **Button** odpowiednio do rozpoczynania odtwarzania, pauzowania i zatrzymywania odtwarzania.

ZADANIE

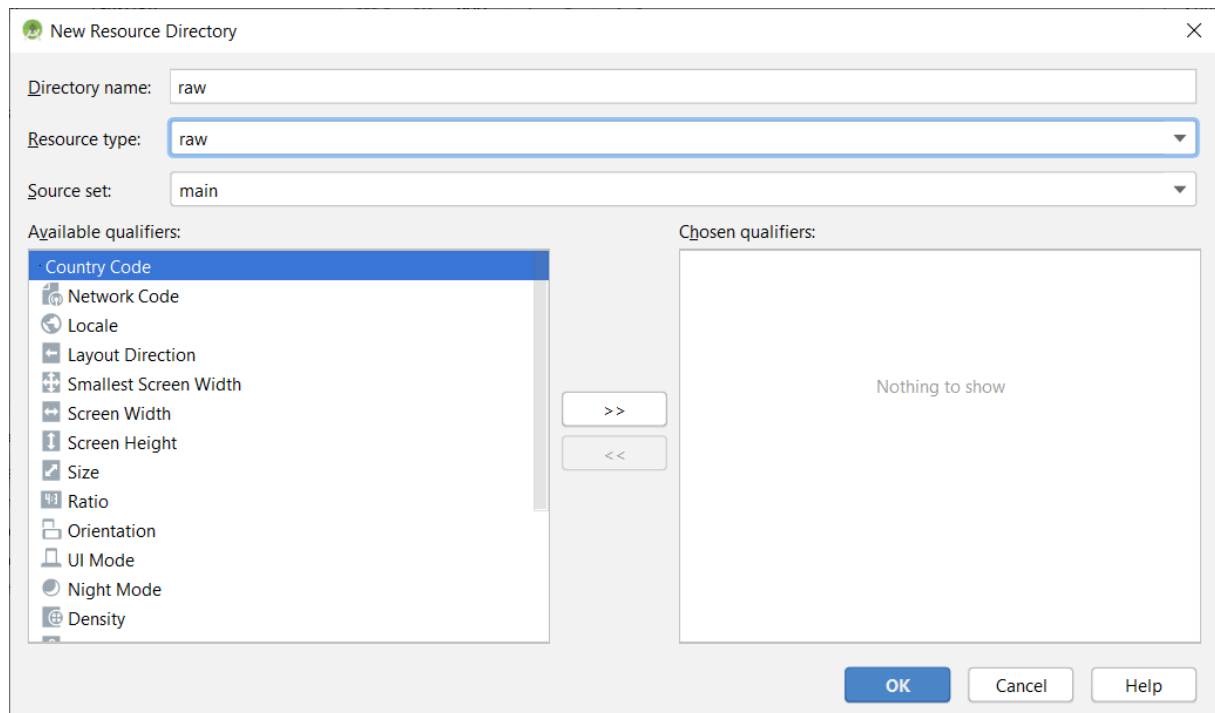
Zmodyfikuj layout **activity_music.xml** wg poniższych obrazów.



Docelowo nasz player będzie odtwarzał plik, który będzie znajdował się w jednym z katalogów aplikacji. Utwórzmy zatem potrzebny folder i umieśćmy w nim plik dźwiękowy.

ZADANIE

Utwórz nowy katalog **raw** w naszym projekcie. W tym celu kliknij PPM na folder **res** i wybierz **new/Android Resource Directory**. Następnie wypełnij pola kreatora tworzenia katalogu wg poniższego rysunku.



Dalej dodaj do utworzonego katalogu plik dźwiękowy.

Możemy teraz przystąpić do oprogramowania kontrolek naszego playera. Na początek utworzymy zmienną klasy **MediaPlayer** i zajmiemy się przyciskiem **Play**.

ZADANIE

W klasie głównej aktywności utwórz zmienną klasy **MediaPlayer**.

```
class MainActivity : AppCompatActivity() {  
  
    private var mediaPlayer: MediaPlayer? = null
```

Następnie zaimplementuj funkcję do odtwarzania dźwięku.

```
fun play() {  
    if (mediaPlayer == null) {  
        mediaPlayer = MediaPlayer.create(context = this, R.raw.best_of_2019_mix)  
        mediaPlayer?.start() // rozpoczęcie odtwarzania  
    }  
}
```

Oprogramuj przycisk **Play** wywołując utworzoną funkcję **play()**.


```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_music)

    play_BT.setOnClickListener { it: View!
        play()
    }
}

```

Uruchom aplikację i sprawdź działanie playera.

W metodzie **play()** za pomocą konstrukcji **MediaPlayer.create()** tworzymy obiekt odpowiedzialny za odtwarzanie dźwięku. Jako parametry podajemy kontekst oraz plik, który ma zostać odtworzony. Dalej rozpoczynamy odtwarzanie dźwięku.

Utwórzmy teraz pozostałe funkcje niezbędne w naszym odtwarzaczu oraz wywołajmy je dla akcji przycisków **Pause** i **Stop**.

ZADANIE

Zaimplementuj kod poniższych metod.

```

fun pause() {
    if (mediaPlayer != null)
        mediaPlayer?.pause()
}

fun stop() {
    if (mediaPlayer != null) {
        mediaPlayer?.release() // zwolnienie zasobu (pliku) powiazanego z playerem
        mediaPlayer = null
    }
}

```

Dopisz również kod przycisków do pauzowania i zatrzymywania odtwarzania.

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_music)

    play_BT.setOnClickListener { it: View!
        play()
        mPlayer_TV.setText("Playing: "+getResources().getResourceEntryName(R.raw.best_of_2019_mix))
    }
    pause_BT.setOnClickListener { it: View!
        pause()
        mPlayer_TV.setText("Paused")
    }
    stop_BT.setOnClickListener { it: View!
        stop()
        mPlayer_TV.setText("Stopped")
    }
}

```

W funkcji **stop()** zwalniamy najpierw zasoby powiązane z playerem, po czym „kasujemy” sam player przypisując do niego wartość **null**.

Z kolei w kodzie przycisków dodane zostały linie, które odpowiadają za wyświetlenie nazwy odtwarzanego pliku (przycisk **Play**) oraz informacji o stanie playera.

Zadbajmy jeszcze o zwolnienie zasobów playera w momencie zatrzymania odtwarzania z powodu zakończenia pliku. W tym celu wprowadźmy modyfikację metody **play()**. Zatrzymajmy także odtwarzanie dźwięku przy wyjściu z aktywności i zwolnijmy zasoby.

ZADANIE

Zmodyfikuj kod metody **play()**.

```
fun play() {
    if(mediaPlayer==null) {
        mediaPlayer = MediaPlayer.create(context: this, R.raw.best_of_2019_mix)
        mediaPlayer?.setOnCompletionListener(MediaPlayer.OnCompletionListener { it: MediaPlayer!
            fun onCompletion(mediaPlayer: MediaPlayer) {
                stop()
            }
        })
    }
    mediaPlayer?.start() // rozpoczęcie odtwarzania
}
```

Dodaj funkcję **onStop()**.

```
override fun onStop() {
    super.onStop()
    stop()
}
```

Przetestuj działanie playera.

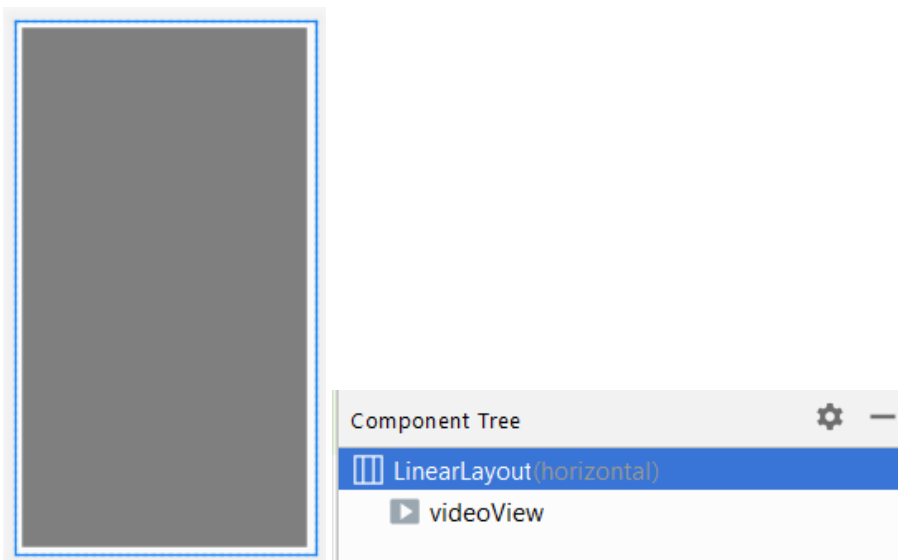
Nasz player jest gotowy.

Odtwarzanie filmów

Przed nami ostatnia aktywność – player video. Rozpoczynamy od utworzenia layoutu.

ZADANIE

Zmodyfikuj layout **activity_video.xml** wg poniższych obrazów.



Ustaw dla komponentu **videoView** następujące parametry:

```
<VideoView
    android:id="@+id/videoView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="10dp" />
```

Potrzebujemy teraz pliku video, który będziemy odtwarzać. W naszej aplikacji używać będziemy plików w formacie **.mp4**. UWAGA: Nazwa pliku może zawierać wyłącznie małe litery, cyfry i znaki podkreślenia.

ZADANIE

Dodaj do pakietu **raw** plik video w formacie mp4.

Przystępujemy do implementacji funkcjonalności odtwarzania filmów. Aby odtwarzać plik video wystarczy dla komponentu **VideoView** ustawić zasób (podać plik), który ma być odtwarzany oraz wywołać metodę **start()**. My jednak chcemy nie tylko móc rozpocząć odtwarzanie filmu ale również jego pauzowanie i zatrzymywanie. Dlatego dla **VideoView** ustawimy również **MediaController**.

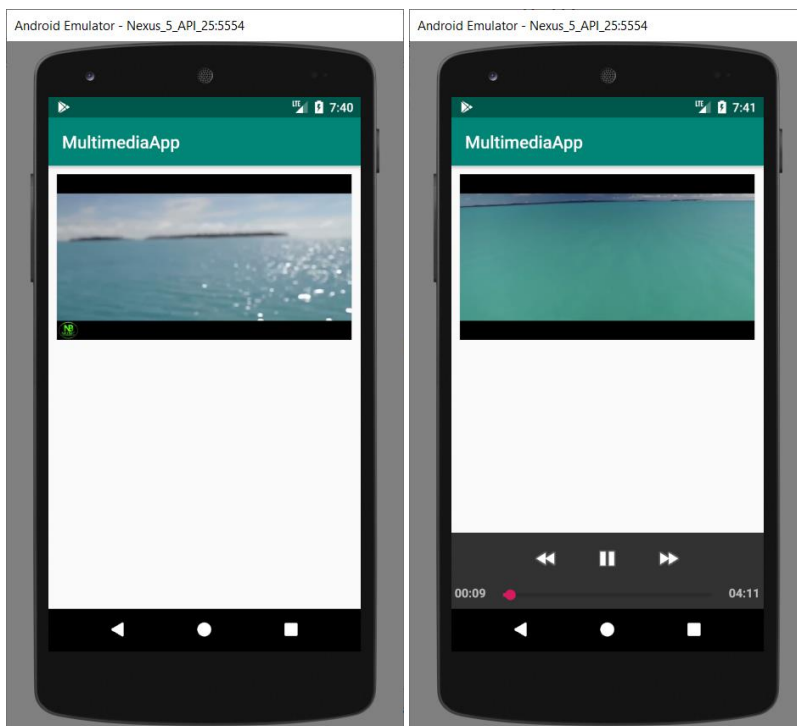
ZADANIE

Dodaj poniższy kod w metodzie **onCreate()** w **VideoActivity**.

```
val path :String = "android.resource://" + packageName + "/" + R.raw.vid_001
val uri: Uri = Uri.parse(path)
videoView.setVideoURI(uri)
videoView.setMediaController(MediaController(context, this))
```

Przetestuj działanie playera video.

Po wywołaniu aktywności **VideoActivity** naszym oczom ukazuje się komponent **VideoView**, który jest czarny – film nie jest odtwarzany. Dopiero po kliknięciu na ten komponent w dolnej części ekranu widzimy **MediaController** - pasek z przyciskami umożliwiającymi sterowanie odtwarzaniem.



Nasz player działa prawidłowo, jednak położenie **MediaControllera** sprawia, że użytkowanie odtwarzacza jest uciążliwe. Dlatego wprowadzimy modyfikację layoutu oraz drobne zmiany w kodzie aktywności.

ZADANIE

Wprowadź poniższe zmiany w layoutcie aktywności video playera.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".VideoActivity">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp">
        <VideoView
            android:id="@+id/videoView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content" />
        </FrameLayout>
    </LinearLayout>
```

Ponadto dla **LinearLayout** ustaw wartość parametru **gravity** na **center**.

Zmodyfikuj również kod **VideoActivity**.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_video)

    val path :String = "android.resource://" + packageName + "/" + R.raw.vid_001
    val uri: Uri = Uri.parse(path)
    videoView.setVideoURI(uri)
    val mediaController = MediaController(context: this)
    videoView.setMediaController(mediaController)
    mediaController.setAnchorView(videoView)
}
```

Teraz nasz video player jest gotowy.

