Układy kontrolki

PUM

Różne szablony aplikacji

Zmiana wyglądu komponentu

- Zmian można dokonać w oknie właściwości, znaczenia poszczególnych parametrów można się łatwo domyślić, ale nie wszystkich
  - np. aby zmienić kolor przycisku (w domyślnym stylu i motywie) trzeba zmienić parametr `backgroundTint` i nie zmieniać parametru `background`
- Zmiana kolorów motywu możliwa w pliku res/values/colors.xml

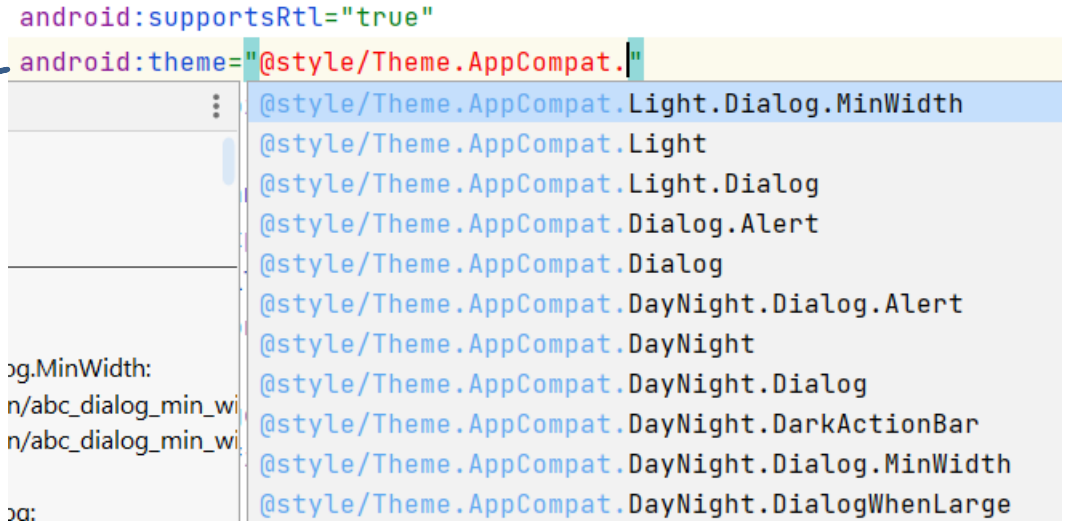
Własne style

- Jeśli chcemy dodać nowy styl na podstawie dotychczasowego można:
  - Zmienić lub nie wygląd jakiegoś komponentu we właściwościach
  - Kliknąć na komponent prawym klawiszem myszki i wybrać z Menu kontekstowego Refactor/ExtractStyle
  - Nadać stylowi nazwę i kliknąć OK
  - Nowy styl zostanie zapisany w pliku zasobów `res/values/styles.xml` i przypisany do komponentu, z którego pochodził – jeśli pliku styles.xml nie było, to zostanie utworzony
  - Teraz nowy styl możemy przypisać innym komponentom szczególnie tego samego typu
  - Możemy też zmieniać nowy styl w pliku `styles.xml`, ale łatwiej jest to zrobić przed zapisaniem w oknie właściwości.

Motywy

- Motyw (Theme) jest okreslany w pliku manifestu, domyślnie jest to plik w zasobach naszej aplikacji, np. android:theme="@style/Theme.Nazwa_aplikacji"
- Motyw określa kolory komponentów naszej aplikacji

Wybór motywu
w pliku manifestu



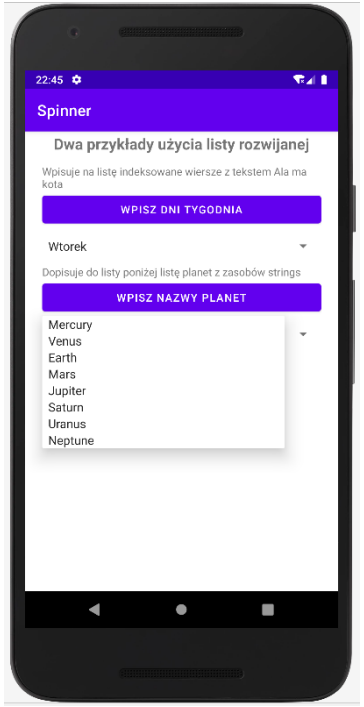
Przykładowe dostępne motywy

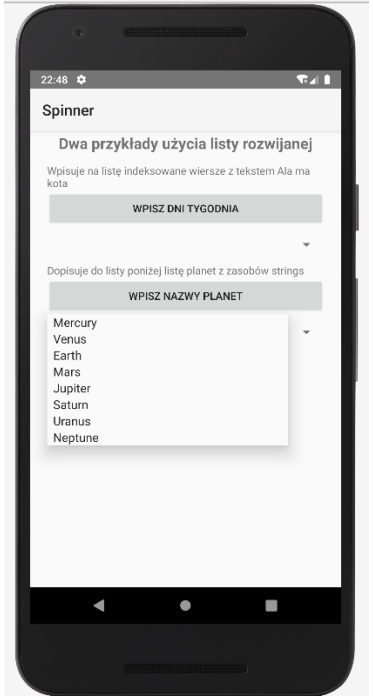
"@style/Theme.AppCompat.DayNight"

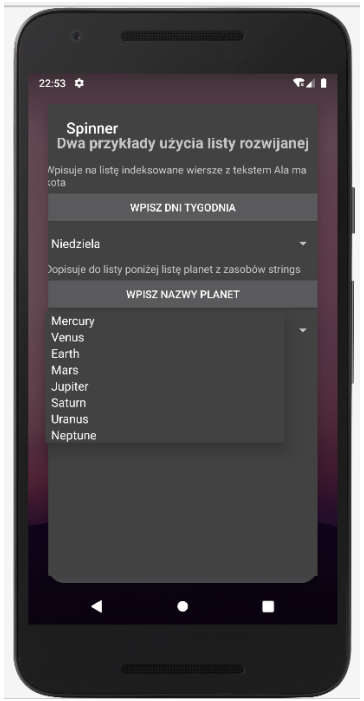
"@style/Theme.MaterialComponents.DayNight.DarkActionBar"

Motyw domyślny

"@style/Theme.AppCompat.Dialog.Alert"







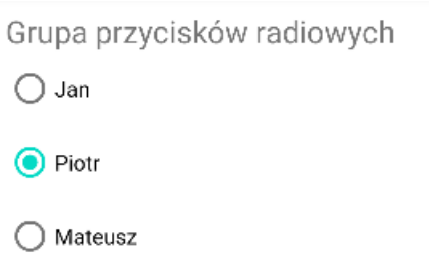
Własny motyw

Domyślny plik motywu, określa kolory aplikacji, można go zamienić w zależności od upodobania

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Spinner" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/purple_500</item>
        <item name="colorPrimaryVariant">@color/purple_700</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
        <!-- Customize your theme here. -->
    </style>
</resources>
```

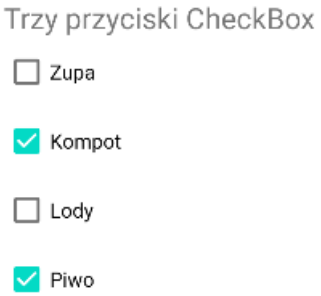
KONTROLKI LIST

Przyciski radiowe



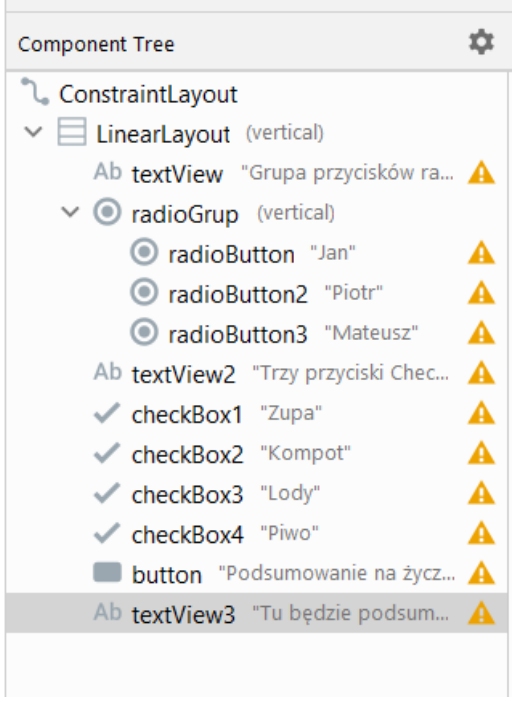
- Pole typu RadioButton (jakCheckBox) jest dwustanowym przyciskiem, który może być *zaznaczony bądź nie.*
- Zwykle pola tego typu są łączone w kontener o nazwie RadioGroup. Zastosowanie tego kontenera powoduje, że pola wyboru zachowują się jako wzajemnie wykluczające się selektory. Oznacza to, że zmiana jednego z przycisków powoduje odznaczenie pozostałych.
- Właściwości dotyczących kroju, stylu czy koloru czcionki zarządzane są podobnie jak w przypadku etykiety TextView.
- Można wywołać metodę `isChecked()` *by sprawdzić czy poszczególny przycisk jest zaznaczony, natomiast metoda* `toggle()`*odpowiada za zmianę jego stanu.*

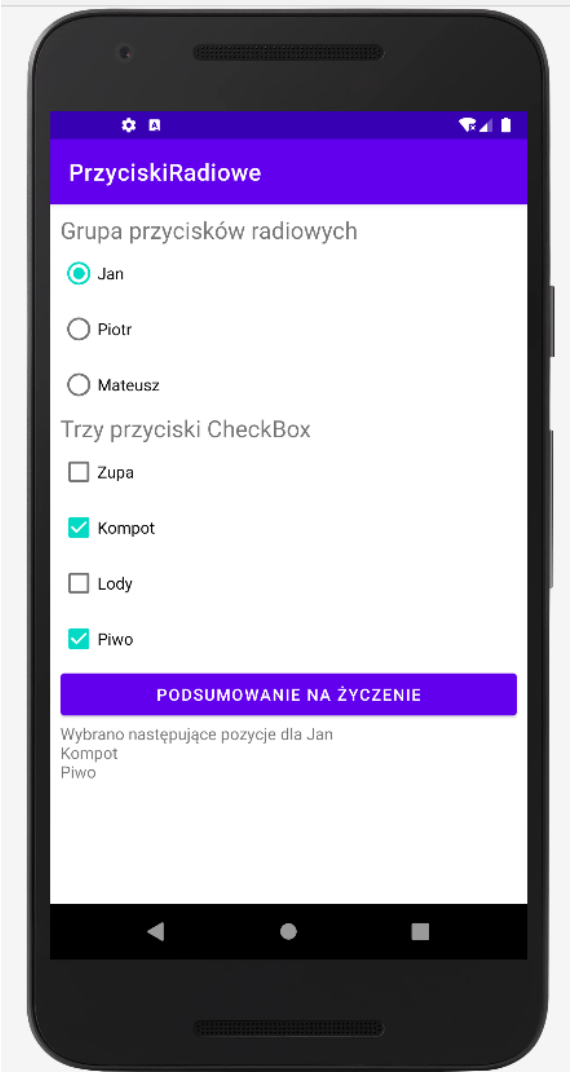
Przyciski CheckBox



- Pole typu CheckBox można interpretować jako dwustanowy przycisk, który może być zaznaczony bądź nie.
- Ekran może zawierać wiele niezależnie działających pól wyboru. W danym czasie więcej niż jeden CheckBox może zostać zaznaczony.
- W przykładowym programie, po kliknięciu przycisku Podsumowanie na życzenie na ekranie pojawi się komunikat o wybranych (zaznaczonych) przyciskach radiowych i CheckBox

Przykład - układ





Wyświetlanie komunikatów o zmianie stanu grupy przycisków radiowych

Po kliknięciu któregoś z przycisków radiowych z grupy w toście pojawia się informacja o zmianie wyboru. Metoda `setOnCheckedChangeListener` wiąże zdarzenie zmiany stanu grupy przycinków radiowych z metodą jego obsługi

Parametr i określa identyfikator wybranego przycisku, wybrany może być tylko jeden

```
RadioGroup radioGroupX = findViewById(R.id.radioGrup);
radioGroupX.setOnCheckedChangeListener((radioGroup, i) -> Toast.makeText(this,
        "Wybrałeś pozycję: " + ((RadioButton)findViewById(i)).getText(),
        Toast.LENGTH_LONG).show());
```

Wyświetlanie komunikatów o zmianie stanu przycisków CheckBox

Po kliknięciu któregoś z przycisków CheckBox w toście pojawia się informacja o zmianie wyboru

Kod dodania nasłuchiwaczy zdarzeń kliknięcia przycisków, można go umieścić w metodzie onCreate() aktywności

```
CheckBox checkBox1 = (CheckBox) findViewById(R.id.checkBox1);
checkBox1.setOnClickListener(this::onClick);
CheckBox checkBox2 = (CheckBox) findViewById(R.id.checkBox2);
checkBox2.setOnClickListener(this::onClick);
CheckBox checkBox3 = (CheckBox) findViewById(R.id.checkBox3);
checkBox3.setOnClickListener(this::onClick);
CheckBox checkBox4 = (CheckBox) findViewById(R.id.checkBox4);
checkBox4.setOnClickListener(this::onClick);
```

Referencja do metody obsługi zdarzenia, jednakowa dla wszystkich przycisków

Kod metody obsługi zdarzenia

```
private void onClick(View view) {
    if (((CheckBox) view).isChecked())
        Toast.makeText(this, "Wybrałeś " + ((CheckBox) view).getText(), Toast.LENGTH_LONG).show();
    else
        Toast.makeText(this, "Odrzuciłeś " + ((CheckBox) view).getText(), Toast.LENGTH_LONG).show();
}
```

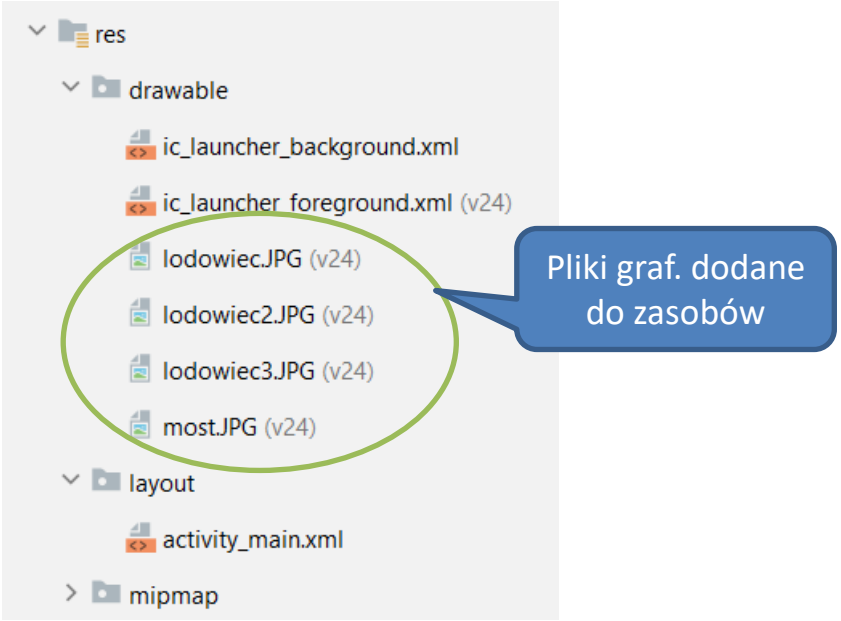
Sprawdzenie stanu wszystkich przycisków po kliknięciu przycisku

Wydruk w kontrolce *textView3*

```
((Button)findViewById(R.id.button)).setOnClickListener(view -> {
    String msg = "Wybrano następujące pozycje dla ";

    msg+= ((RadioButton)findViewById(radioGroupX.getCheckedRadioButtonId())).getText();

    if(checkBox1.isChecked()) msg += "\n" + checkBox1.getText();
    if(checkBox2.isChecked()) msg += "\n" + checkBox2.getText();
    if(checkBox3.isChecked()) msg += "\n" + checkBox3.getText();
    if(checkBox4.isChecked()) msg += "\n" + checkBox4.getText();

    ((TextView) findViewById(R.id.textView3)).setText(msg);
});
```

Obrazki ImageView i ImageButton

- ImageView i ImageButton pozwalają na umieszczenie grafik (gif, jpg, png, …).
- Przed dodaniem obrazka należy plik z grafiką umieścić w zasobach w folderze res/drawable.
- Następnie można kontrolki ImageView i ImageButton umieszczać na ekranie podając położenie pliku grafiki w zasobach.
- Położenie pliku grafiki to atrybut (właściwość) src, jeśli atrybutu src nie ma w oknie Attributes w widoku Design, to należy wartość atrybutu wpisać w widoku Code.
- Duże pliki graficzne przed dodaniem do zasobów dobrze jest „odchudzić”.
- Dla obu kontrolek można zdefiniować metodę obsługi zdarzenia onClick.

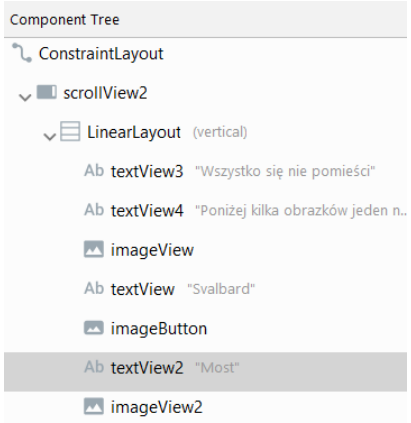


Brak miejsca na ekranie – możliwe rozwiązania

- Przewijanie na ekranie – ScrollView
- Listy rozwijane
- Okna dialogowe
- Nowa dodatkowa aktywność do wprowadzania danych
- Fragmenty

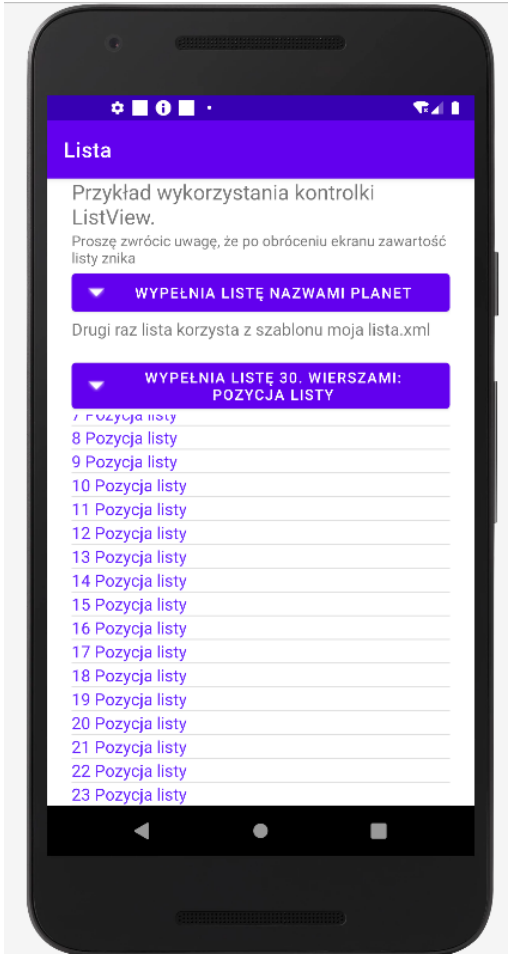
ScrollView

- Komponent ScrollView jest przydatny w sytuacjach, gdy do wyświetlenia jest więcej danych niż można pomieścić na ekranie.
- ScrollView zapewnia dostęp do danych z użyciem pasków przewijania.
- Część danych jest wyświetlana, pozostała część, która nie mieści się na ekranie, jest ukryta.
- Przewijanie może być pionowe albo poziome.



ListView

- Obecnie rzadko używana kontrolka, bo zajmuje dużo miejsca na ekranie.
- Ma własny mechanizm przewijania, gdy wszystkie pozycje nie mieszczą się na ekranie.
- Nadaje się dobrze do zastosowania na specjalnie dodanej nowej aktywności, która po dokonaniu wyboru znika.
- Gdy brak miejsca i nie chcemy dodawać do aplikacji nowej aktywności dobrze jest stosować listę rozwijaną Spinner.
- Komponent ListView jest obecnie uznawany za przestarzały.



ListView

Dane do wypełnienia listy mogą pochodzić z różnych źródeł.
Do przygotowania zestawu pozycji, które będą wyświetlone w liści wykorzystuje się obiekt klasy ArrayAdapter<T>
Konstruktor obiektu pobiera dane z różnych źródeł, mogą to być zasoby, mogą być dane wpisane w kodzie aplikacji, jak w poniższym przykładzie lub dane z bazy danych czy internetu.

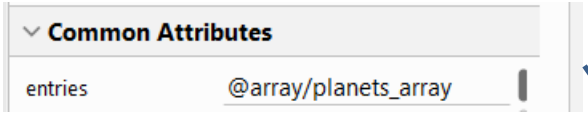
Tablica danych listy

```
String[] items = {"Niedziela", "Poniedziałek", "Wtorek", "Środa",
        "Czwartek","Piatek", "Sobota" };
ArrayAdapter<String> adapter= new ArrayAdapter<String>(this,
        android.R.layout.simple_list_item_1, items);
```

Podstawowy układ listy, dostępny w zasobach Androida, można zastosować własny

ListView – dane z zasobów

Gdy pozycje listy są w zasobach,
można nie korzystać z adaptera,
a określić źródło danych w układzie,
jako właściwość `entries` kontrolki listy



W oknie właściwości

```
<string name="app_name">Lista</string>
<string-array name="planets_array">
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
    <item>Jupiter</item>
    <item>Saturn</item>
    <item>Uranus</item>
    <item>Neptune</item>
</string-array>
```

```
<ListView
    android:id="@+id/lista"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:entries="@array/planets_array" />
```

W kodzie XML układu

ListView – dane z zasobów

Dane dla listy można umieścić z pliku strings.xml zasobów, jak w przykładzie, po prawej stronie.
Poniżej kod stworzenia adaptera.

```
<string name="app_name">Lista</string>
<string-array name="planets_array">
    <item>Mercury</item>
    <item>Venus</item>
    <item>Earth</item>
    <item>Mars</item>
    <item>Jupiter</item>
    <item>Saturn</item>
    <item>Uranus</item>
    <item>Neptune</item>
</string-array>
```

```
final ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.planets_array, android.R.layout.simple_list_item_1);
```

Pobranie tablicy z zasobów

Dane dla listy umieszczone w pliku strings.xml zasobów,
można odczytać w kodzie jak w przykładzie poniżej.

```
Resources res = getResources();
String[] planets = res.getStringArray(R.array.planets_array);
```

Pobraną tablicę można wykorzystać do odczytania wybranego elementu lub do stworzenia adaptera.
Dane dla list można też pobierać z bazy danych lub plików tekstowych.

ListView dodanie adaptera i obsługi zdarzenia

Na koniec pozostało dopisanie do kontrolki `listview` klasy `ListView` adaptera i dodanie obsługi zdarzenia wyboru pozycji na liście.

```
ListView listview = (ListView)findViewById(R.id.lista);
listview.setAdapter(adapter);

listview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(getApplicationContext(),
                "Pozycja nr: " + position + " - " + adapter.getItem(position),
                        Toast.LENGTH_LONG).show();
    }
});
```

ListView – własny układ

Na koniec można dodać własny układ kontrolki , np. taki jak poniżej zapisany w pliku res/layout/mojalista.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#651FFF"
    android:textSize="16sp" />
```

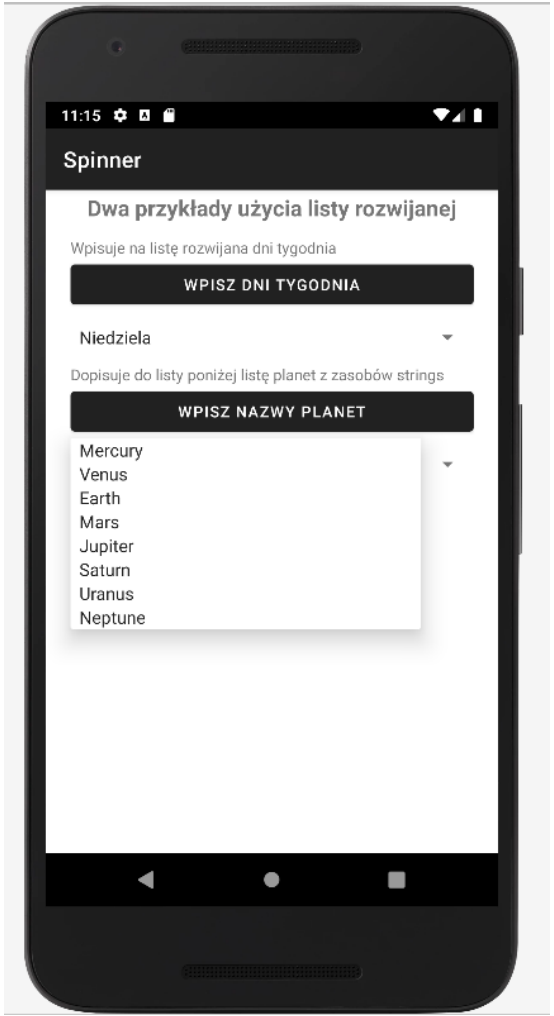
i wykorzystać go tworząc adapter

```
final ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
        R.array.planets_array, R.layout.mojalista);
```

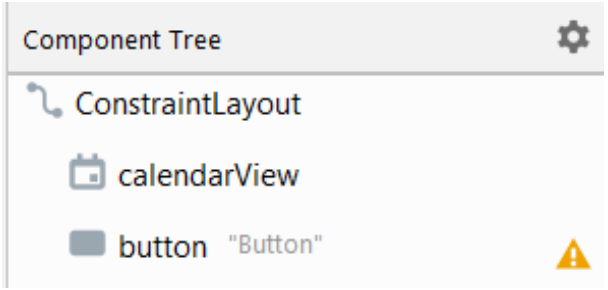
Adres własnego układu

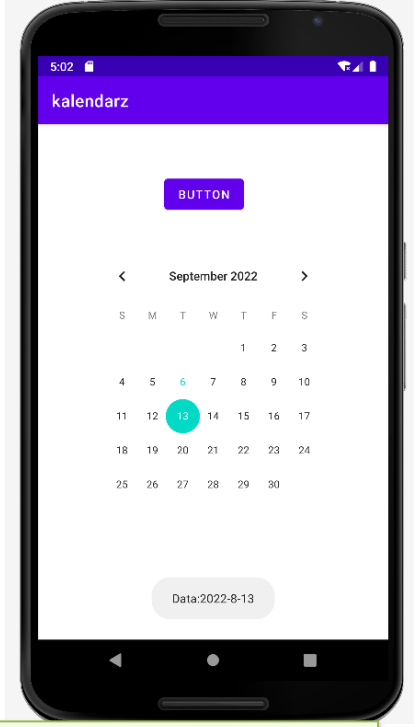
Lista rozwijana – Spinner

- Zajmuje na ekranie mniej miejsca niż ListView
- Działa prawie identycznie
- Po dokonaniu wyboru widoczna jest wyłącznie wybrana pozycja listy
- Wbudowany układ to: `android.R.layout.simple_spinner_item`
- Najczęściej obsługiwane zdarzenia: `setOnItemSelectedListener`, ~~`setOnItemClickListener`~~



Kalendarz





Przykład: po wyborze daty w toście wyświetla się wybrana data.

```
CalendarView calendar = (CalendarView)findViewById(R.id.calendarView);
calendar.setOnDateChangeListener((view, year, month, day) -> {
    Toast.makeText(getApplicationContext(),
                   "Data:" + year + "-" + ++month + "-" + day,
                   Toast.LENGTH_LONG).show();
});
```