Moja pierwsza aplikacja mobilna w Android Studio

Warunki wstępne

1. Smartfon: włączyć *Ustawienia / Opcje programisty:*

✔ Debugowanie USB;

✔ Debugowanie bezprzewodowe (*Wireless Debugging*) / *Pair Device using QR Code (Android > 11, dostęp do Wi-Fi)*

✔ Zezwalaj na instalacje z nieznanych źródeł.

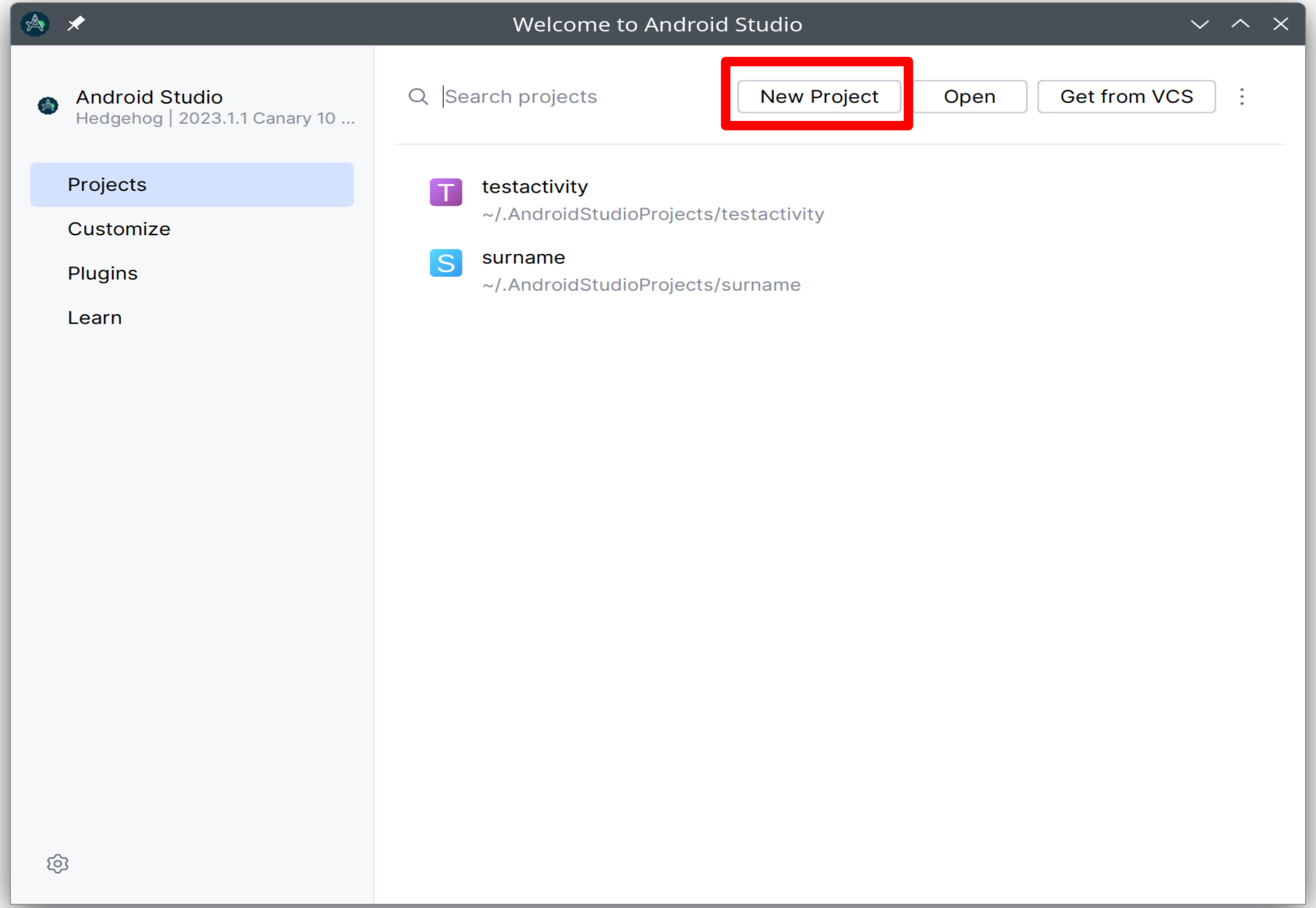
Warunki wstępne

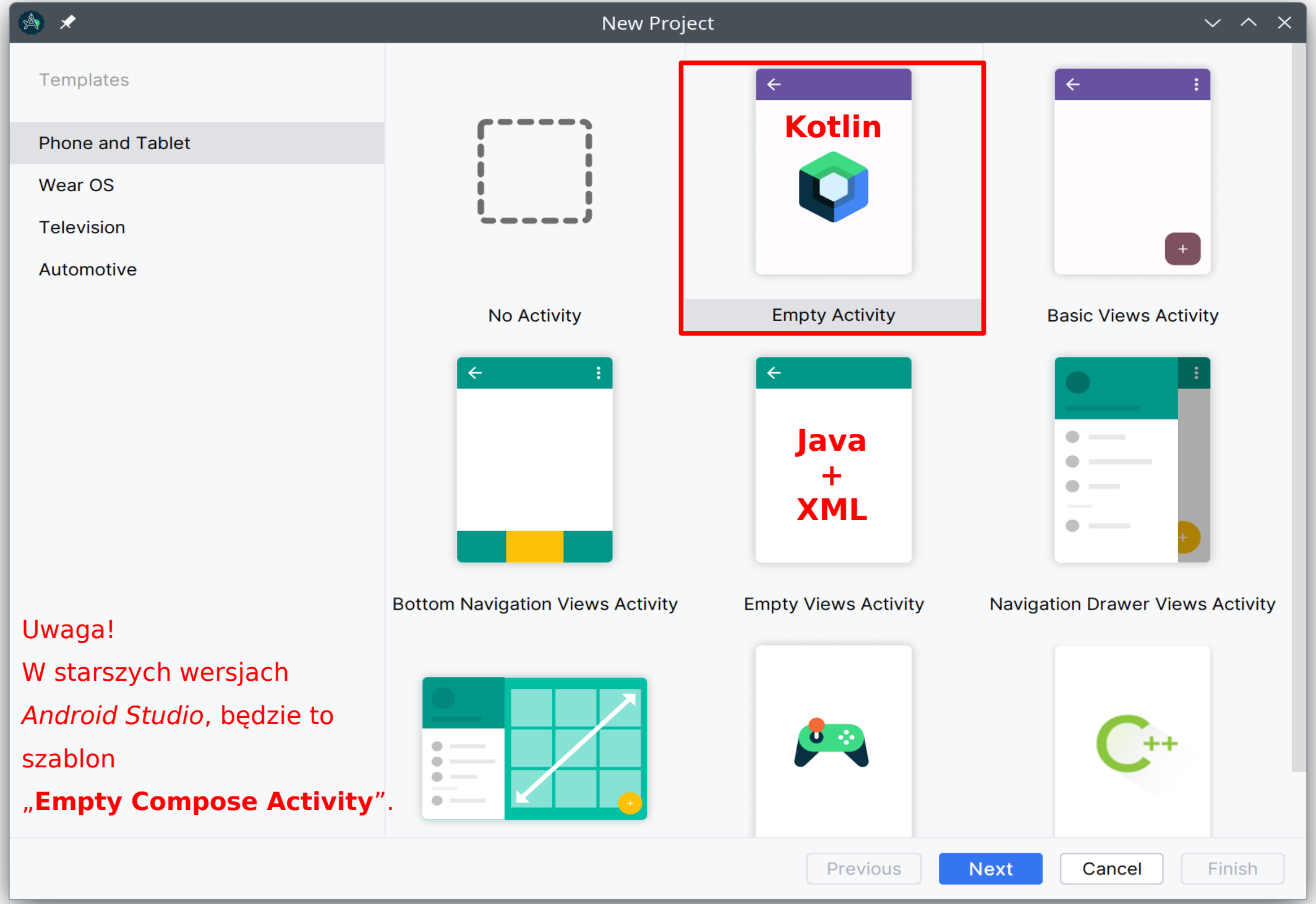
2. BIOS/UEFI: włączyć opcję *Virtual Technology* (VT).

3. Windows: instalacja funkcji *Hyper-V*, oba składniki (*Panel sterowania / Programy i funkcje*)

4. Windows: instalacja *JDK* (lub *openJDK*).

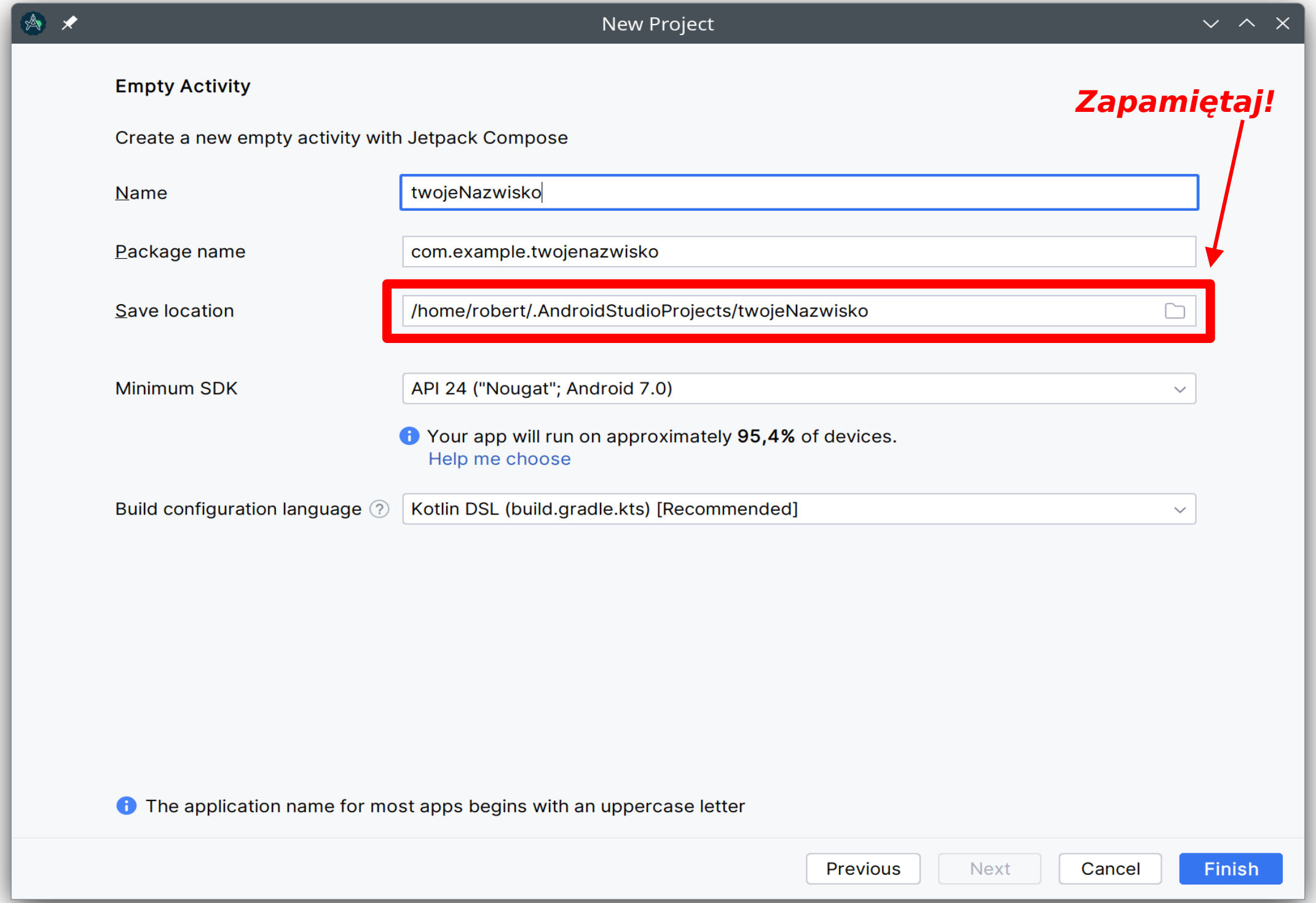
5. Windows: instalacja *Android Studio*.

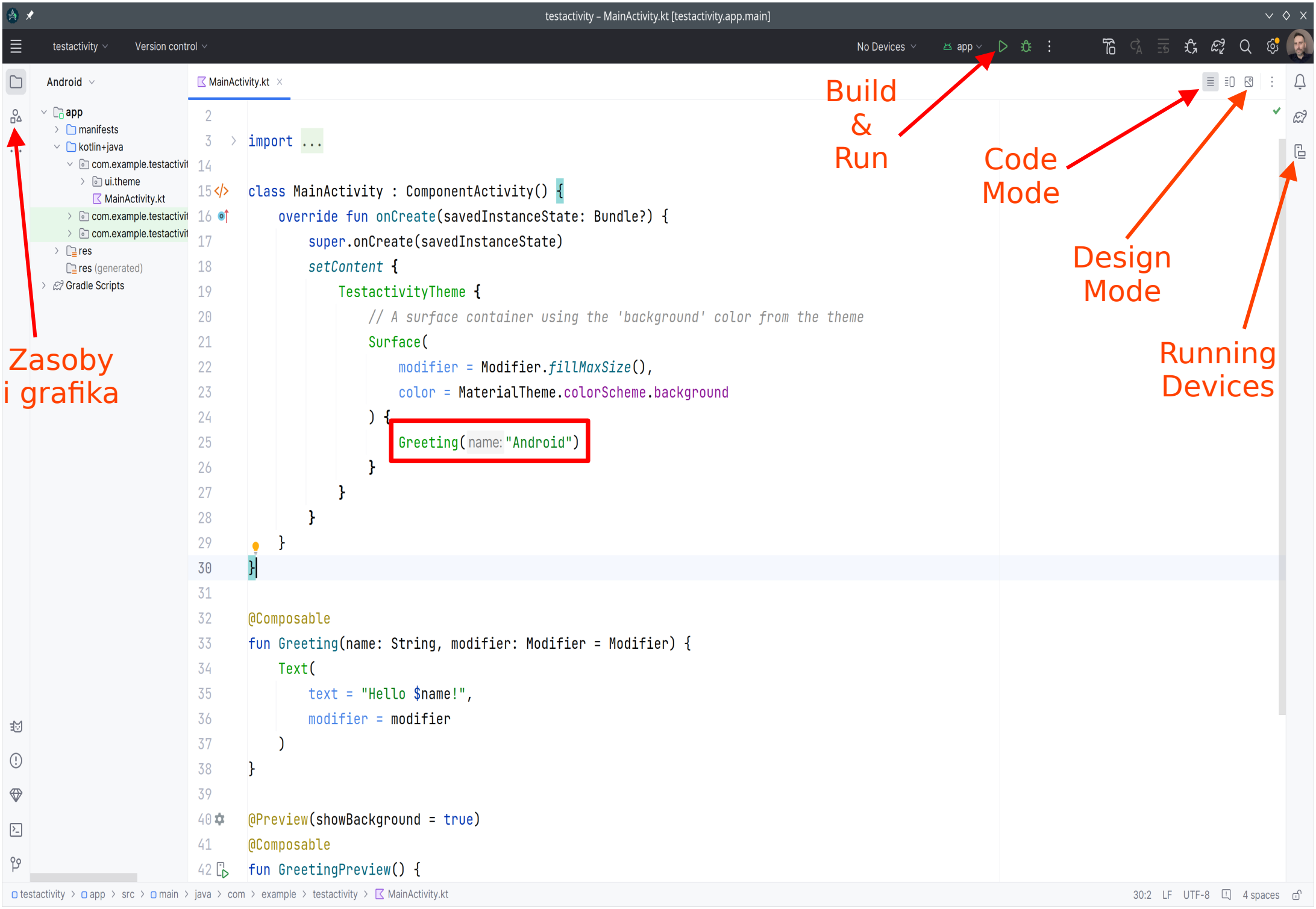

Welcome to Android Studio
Android Studio
Hedgehog | 2023.1.1 Canary 10 ...
Projects
Customize
Plugins
Learn
Search projects
New Project
Open
Get from VCS
testactivity
~/.AndroidStudioProjects/testactivity
surname
~/.AndroidStudioProjects/surname



Uwaga!

W starszych wersjach *Android Studio*, będzie to szablon „**Empty Compose Activity**”.


New Project
Empty Activity
Zapamiętaj!
Create a new empty activity with Jetpack Compose
Name
twojeNazwisko
Package name
com.example.twojenazwisko
Save location
/home/robert/.AndroidStudioProjects/twojeNazwisko
Minimum SDK
API 24 ("Nougat"; Android 7.0)
Your app will run on approximately 95,4% of devices.
Help me choose
Build configuration language
Kotlin DSL (build.gradle.kts) [Recommended]
The application name for most apps begins with an uppercase letter
Previous
Next
Cancel
Finish


testactivity – MainActivity.kt [testactivity.app.main]
testactivity
Version control
No Devices
app
Android
MainActivity.kt
app
manifests
kotlin+java
com.example.testactivit
ui.theme
MainActivity.kt
com.example.testactivit
com.example.testactivit
res
res (generated)
Gradle Scripts
import ...
class MainActivity : ComponentActivity() {
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContent {
TestactivityTheme {
// A surface container using the 'background' color from the theme
Surface(
modifier = Modifier.fillMaxSize(),
color = MaterialTheme.colorScheme.background
) {
Greeting(name: "Android")
}
}
}
}
}
@Composable
fun Greeting(name: String, modifier: Modifier = Modifier) {
Text(
text = "Hello $name!",
modifier = modifier
)
}
@Preview(showBackground = true)
@Composable
fun GreetingPreview() {
Build & Run
Code Mode
Design Mode
Running Devices
Zasoby i grafika
testactivity › app › src › main › java › com › example › testactivity › MainActivity.kt
30:2
LF
UTF-8
4 spaces

Emulatory smartfonów

Działające, przetestowane:

- **Nexus S** - API 31 / 33
- **Pixel** - API 28 / 30 / 34
- **Pixel XL** - API 28 / 29
- **Pixel 2*** - API 28 / 30 / 33 / 34
- **Pixel 3** - API 30
- **Pixel 3a** - API 28 / 33 / 34
- **Pixel 4** - API 30 / 33 / 34
- **Pixel 4 XL** - API 28 / 33
- **Pixel 5** - API 30 / 33 / 34
- **Pixel 6** - API 28 / 30 / 33 / 34
- **Pixel 6 Pro** - API 30 / 33 / 34
- **Pixel 7** - API 30 / 33
- **Pixel 7 Pro*** - API 30 / 34

* Posiada działający przycisk WSTECZ.

Emulatory smartfonów

- API do emulatora powinna „ważyć” ok. **8**-**10**GB. Jeśli mniej - to coś jest nie tak.

- Uruchamiamy tylko jeden emulator w tym samym czasie. W zakładce „*Running Devices*” sprawdzamy, czy poprzednio uruchamiany emulator został zamknięty.

- Zielona strzałka ► - buduje projekt, wysyła go na smartfon i uruchamia go na smartfonie.

- Gdy wyskoczy okienko z informacją o debuggerze - zamykamy je (*Cancel*).

- Na uruchomienie aplikacji na smartfonie czekamy maksymalnie 5 minut.

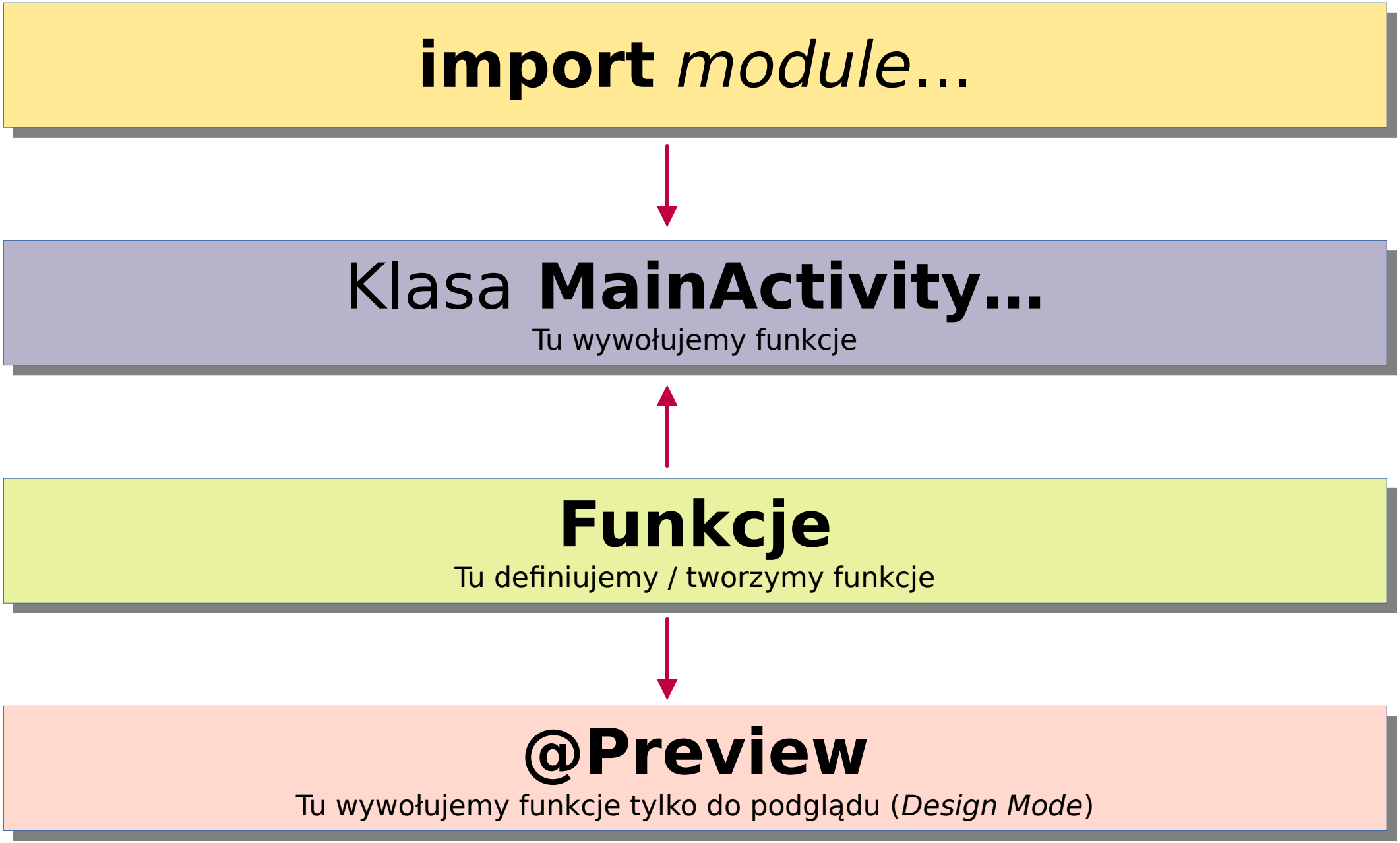
Jetpack Compose

Narzędzie do budowania UI w Kotlinie, zastępuje *Android View System* (Java + XML)

@Composable = element do złożenia (klocek)

- mniejsza ilość kodu
- działa tylko w Kotlinie
- bardziej logiczny / przejrzysty
- jeden plik *.kt (XML nie jest konieczny)
- w przypadku zmiany danych i stanu jakiegoś elementu, rekompozycji ulega tylko zmieniony element
- wykorzystuje wiele procesorów / wątków w tym samym czasie
- funkcja *Preview* oszczędza czas (możemy tworzyć wiele sekcji *Preview*)

Filozofia *JetPack Compose*



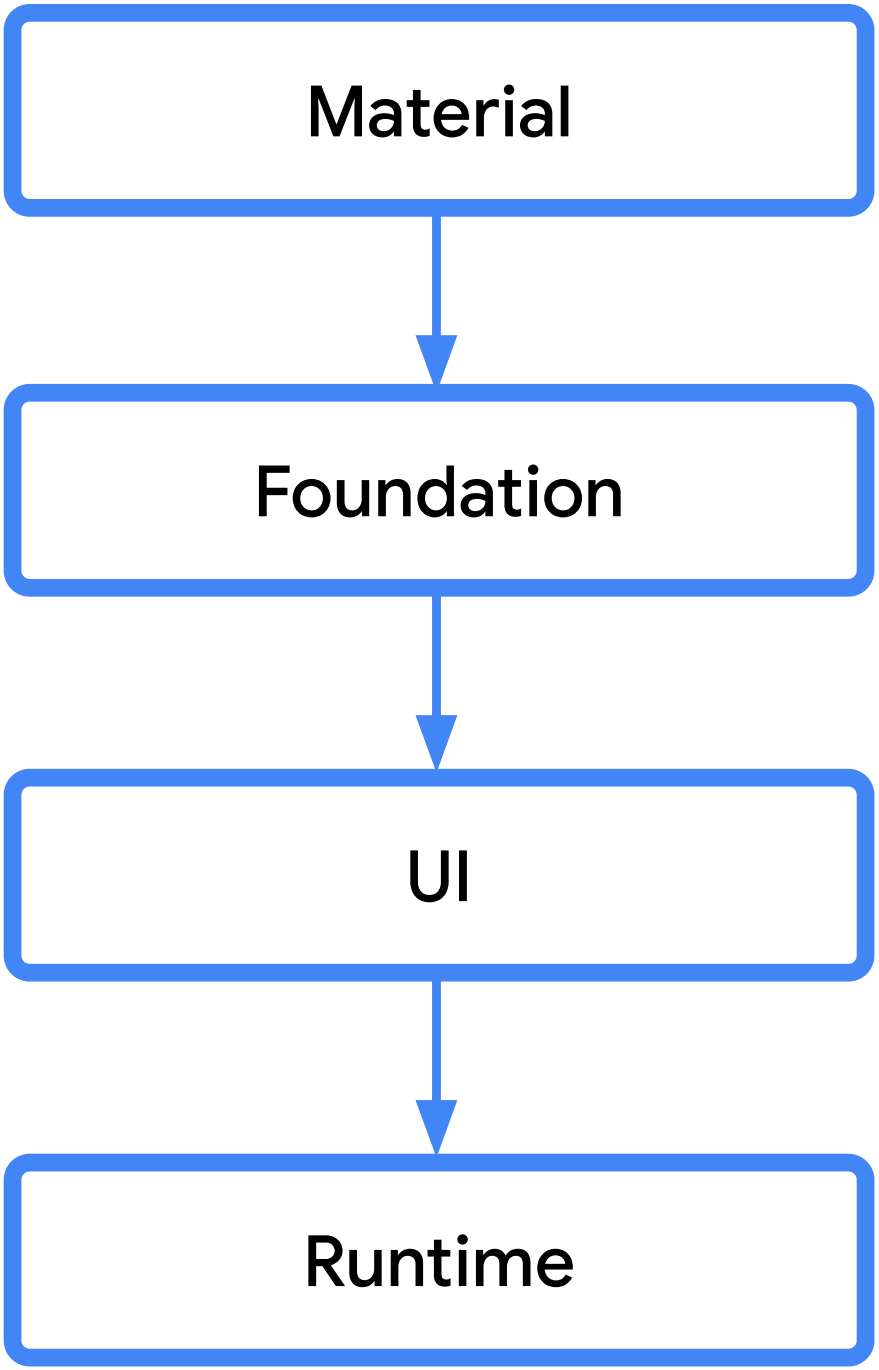
Główne warstwy Jatpack Compose

Implementacja *Material Design*, szablony, style, ikony, przyciski

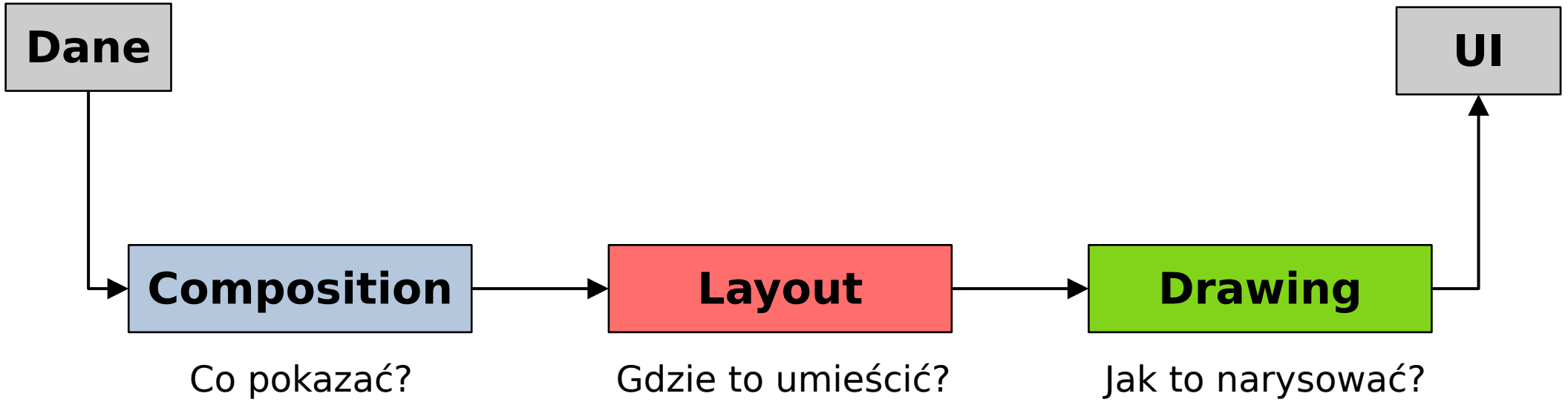
Bloki: Column, Row, obsługa gestów, budowa własnego design'u

Moduły: ui-text, ui-graphics, ui-tooling, LayoutNode, Modifier

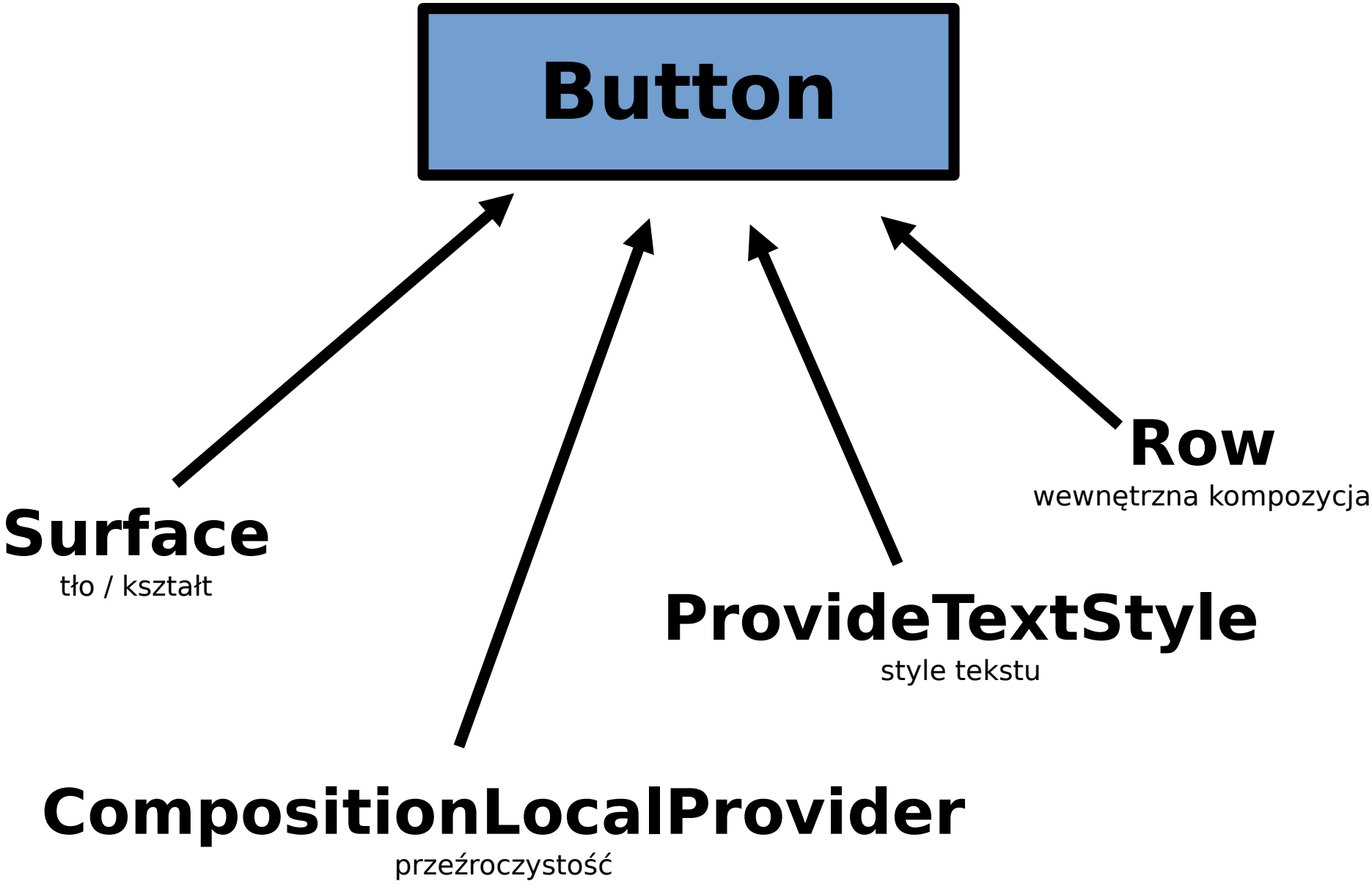
Komponenty: remember, mutableStateOf, @Composable, SideEffect...



Etapy kompozycji



Elementy mogą zawierać inne elementy w różnych konfiguracjach:



Typowy standard budowania funkcji

```
fun nazwaFunkcji(parX, parY) {
    instrukcje; // Średnik jest dopuszczalny, ale nadmiarowy
}
```

```
// Funkcja bez parametrów:
fun nazwaFunkcji() {
    instrukcje
}
```

```
// Także funkcja bez obsługi parametrów:
fun nazwaFunkcji {
    instrukcje
}
```

Elementy aplikacji mobilnej

Composables (czyt. *kompozeboły*) **- elementy do złożenia**

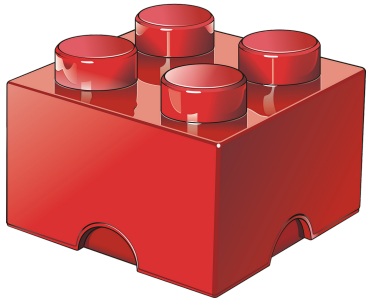
- **Text** - wyświetlany tekst
- **TextField** - wpisywany tekst
- **Button** - przycisk (może być *Filled*, *Tonal*, *Outlined*, *Elevated*); istnieją także przyciski typu floating, np. *FloatingActionButton*;
- **Chip** - rodzaj buttona z predefiniowaną ikoną; rodzaje: *AssistChip*, *FilterChip*, *InputChip*, *SuggestionChip*;
- **Image** - obraz (*.png, *.jpg)
- **Row** - element do rozmieszczania innych elementów poziomo; może być elementem potomnym wobec kolumny
- **Column** - element do rozmieszczania innych elementów pionowo
- **Divider** - linia oddzielająca (odpowiednik <*hr*>)
- **Spacer** - pusty element oddzielający dwa inne elementy (np. pusta linia)

Elementy aplikacji mobilnej

Composables (czyt. *kompozeboły*) **- elementy do złożenia**

- **Card** / **ElevatedCard** / **OutlinedCard** - analogia do ; mogą w sobie zawierać kolumny i wiersze; są o poziom wyżej niż tło; artybutem obowiązkowym jest opis (coś jak „alt”); używać do: podgląd produktu w sklepie, podgląd news’a (uwypukla zawartość spośród innych); nie posiadają możliwości scrollowania ani zamykania (chyba że są w innym elemencie o takich właściwościach);
- **Surface** - płaszczyzna na kompozycje, kontrola nad pikselami, animacje, gry; może być elementem innych elementów (np. **Button**) i dostarczać im tła, kształtu;
- **Scaffold** - rusztowanie (struktura) całej strony, na którym można umieścić **TopAppBar** (górny pasek narzędziowy), **BottomAppBar** oraz **floatingActionButton**;
- **Dialog** - wyskakujące okienko typu „alert” lub „input”; czeka na reakcję użytkownika;
- **VerticalPager** / **HorizontalPager** - przeglądarka zdjęć
- **FlowRow** / **FlowColumn** - elementy, które nie mieszczą się przechodzą do następnej linii / kolumny (zawijają się).

Modifier



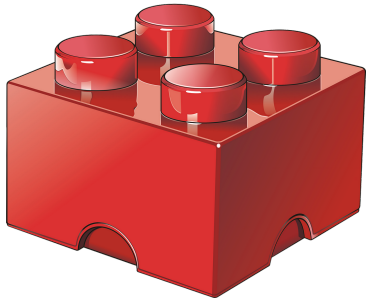
1. Obiekt w języku *Kotlin / JetPack Compose* zarządzający właściwościami elementów kompozytowych i zmieniający je dynamicznie w zależności od interakcji z użytkownikiem.

2. Modyfikatory danego elementu zależą od rodzaju elementu nadrzędnego i potomnego (komunikują się ze sobą).

3. Kolejność atrybutów w Modifier ma znaczenie (wykonywane są według kolejności, a nie wszystkie jednocześnie).

```
Spacer(modifier = Modifier.height(30.dp))
```

Modifier

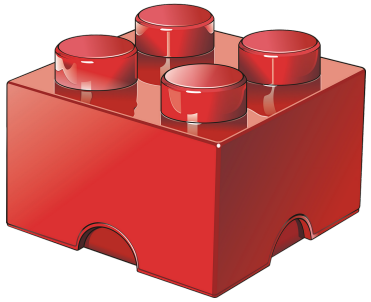


4. Modyfikatory można przypisać do zmiennej i używać wiele razy.

```
val mojModifier = Modifier.padding(12.dp).background(Color.Gray)


@Composable
fun jakakolwiekFunkcja(modifier = mojModifier) {
        // instrukcje
}
```

Modifier

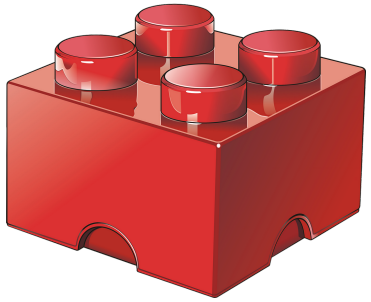


5. Do modyfikatorów będącymi zmienną, można dodawać atrybuty:

```
val mojModifier = Modifier.padding(12.dp).background(Color.Gray)


@Composable
fun jakakolwiekFunkcja(modifier = mojModifier.clickable) {
        // instrukcje
}
```

Modifier



6. Zmienne reprezentujące nasze modyfikatory, możemy traktować jako wartości innych zmiennych modyfikatorów.

```
val mojModifier = Modifier.padding(12.dp).background(Color.Gray)

innyModifier.then(mojModifier)

@Composable

fun jakakolwiekFunkcja(modifier = innyModifier) {

        // instrukcje

}
```

var **zmienna** = **mutableStateOf(***false***)**

Funkcje zawierające te zmienne są **obserwowane** przez *Compose*

(tworzony jest obserwowalny *MutableState*).

Zmiana wartości zmiennej (zmiana jej **stanu**) powoduje ponowne

wywołanie tych funkcji (ich całościową rekompozycję).

„Any time a state is updated a recomposition takes place”.

var **zmienna** = **mutableStateOf(***false***)**

MutableStateOf() to element języka Kotlin i nie jest funkcją kompozytową (może być używana poza *@Composable*, choć nie musi).

Deklarację można napisać też tak (poza @Composable):

var text **by** mutableStateOf("")

by - tworzy link między zmienną a stanem i wymaga:

```
import  androidx.compose.runtime.getValue
import  androidx.compose.runtime.setValue // tylko jeśli deklarujemy var
```

Aby uniknąć rekompozycji niektórych elementów w funkcji, należy umieścić je w **SideEffect{}** (każdy kod niekompozytowy).

var **zmienna** by **remember** {**mutableStateOf(***false***)**}

1. *Remember* zapamiętuje **pierwotną** wartość i przywraca ją podczas rekreakcji (np. zmiana orientacji smartfona).

2. *Remember* zapamiętuje **ostatnią** wartość i przywraca ją podczas rekompozycji.

„*During recomposition, remember returns the value that was **last** stored*”.

3. *Remember* jest funkcją kompozytową używaną **tylko** wewnątrz *@Composable* i przekazuje wartości w głąb elementów.

var **zmienna** by **remember** {**mutableStateOf(***false***)**}

Trzy równoważne formy zapisu:

val zmienna **=** remember { mutableStateOf(false) }

val (zmienna, setValue) **=** remember { mutableStateOf(false) }

var zmienna **by** remember { mutableStateOf(false) }

Ciekawy przykład:

var **nazwaZmiennej** = **remember { 0 }** // Próba zmiany tej zmiennej nie uda się, rekompozycja nie nastąpi

RememberSaveable - ostatnie wartości zmiennej zapisane są w *Bundle*, można je odczytać nawet po rekompozycji, rekreacji (zmiana orientacji) i po awarii.

Układ elementów

Column()

Element 1

Element 2

Element 3

Element 4

Stosować:

- verticalArrangement
- horizontalAlignment

Funkcje zawierające te elementy są „kompozytowe” (wizualne) i powinny być poprzedzone wyrażeniem

@Composable

(jest to informacja dla kompilatora, że dane przeznaczone są do tworzenia UI)

Kontekstem dla nich jest

LocalContext.current

Row()

El. 1 | El. 2 | El. 3

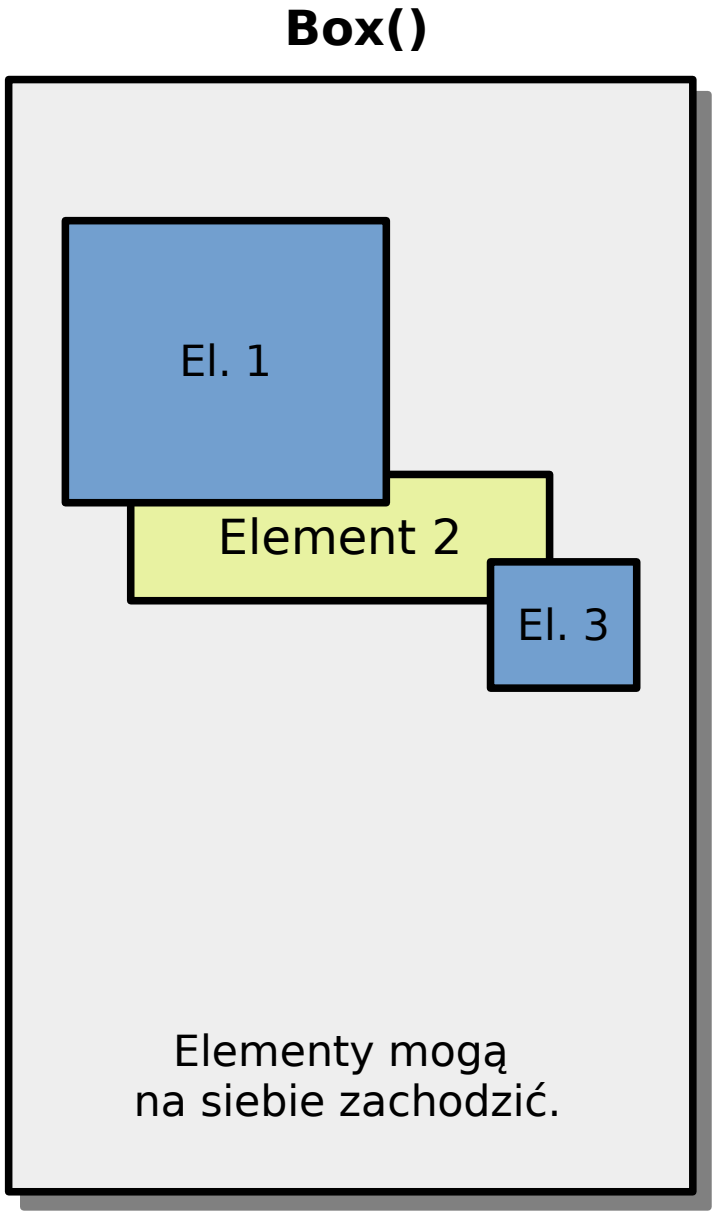
El. 4 | El. 5 | El. 6

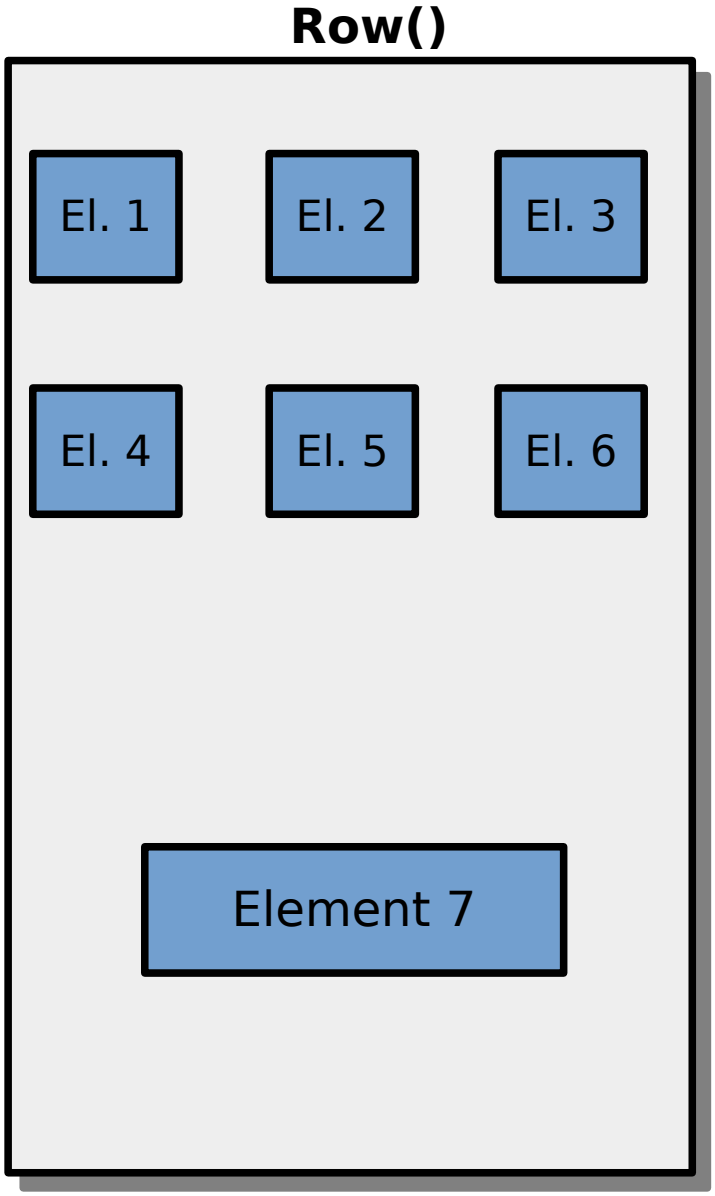
Element 7

Stosować:

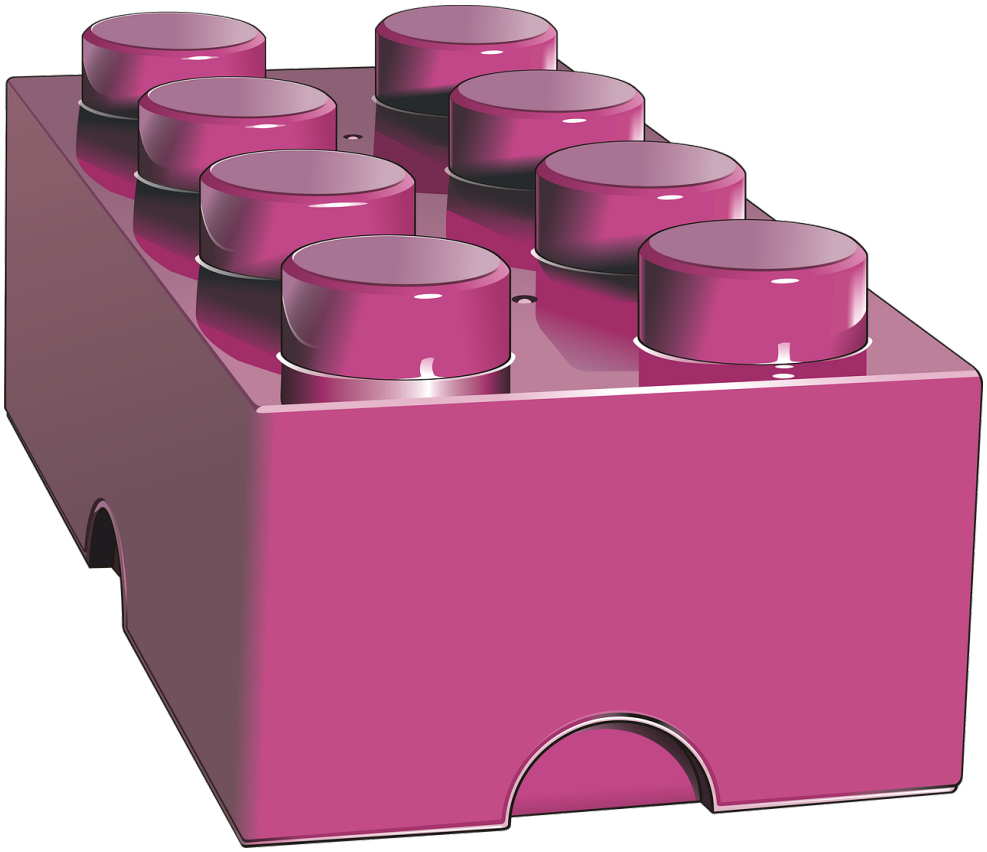
- horizontalArrangement
- verticalAlignment

Układ elementów





Responsywność



```
@Composable
fun Card() {
    BoxWithConstraints {
        if (maxWidth < 400.dp) {
            Column {
                Image(...)
                Title(...)
            }
        } else {
            Row {
                Column {
                    Title(...)
                    Description(...)
                }
                Image(...)
            }
        }
    }
}
```

```
@Composable
fun SpisUczniow() {
    Text("Brajan Sheridan")
    Text("Ariel Nowak")
}
```

Wynik:


```
@Composable
fun SpisUczniow() {
    Column {
        Text("Brajan Sheridan")
        Text("Ariel Nowak")
    }
}
```

Wynik:

```
@Composable
fun Kompozycja() {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Image(painter = painterResource(id = R.drawable.kot),
            contentDescription = "Opiekunowie Mruczka")
        Column {
            Text("Brajan Sheridan")
            Text("Kevin Bronkiewicz")
        }
    }
}
```

Podgląd:

Brajan Sheridan

Kevin Bronkiewicz

Toast wewnątrz MainActivity

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SurnameTheme { // Nazwa tego elementu zależy od nazwy projektu
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {

val wiadomosc = "Hello Kitty!”
val czasWyswietlania = Toast.LENGTH_LONG
val mojToast = Toast.makeText(applicationContext, wiadomosc, czasWyswietlania)

// Wywołanie:
mojToast.show()
                }
            }
        }
    }
}
```

Wynik:

Toast na zewnątrz MainActivity v.1

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SurnameTheme { // Nazwa tego elementu zależy od nazwy projektu
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                        // Wywołanie funkcji i przekazanie jej argumentu:
                        zewnetrznyToast(applicationContext)
                 }
            }
        }
    }
}

@Composable
fun zewnetrznyToast(context:Context) {
    val wiadomosc = "Hello Kitty!"
    val czasWyswietlania = Toast.LENGTH_LONG
    val mojToast = Toast.makeText(context, wiadomosc, czasWyswietlania)
    mojToast.show()
}
```

Wynik:

Toast na zewnątrz MainActivity v.2

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            SurnameTheme { // Nazwa tego elementu zależy od nazwy projektu
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                        // Wywołanie funkcji i przekazanie jej argumentu:
                        wyswietlToast("Hello Kitty!")
                    }
                }
            }
        }
}

@Composable
fun Context.wyswietlToast(wiadomosc: String) {
    Toast.makeText(applicationContext, wiadomosc, Toast.LENGTH_LONG).show()
}
```

Wynik:

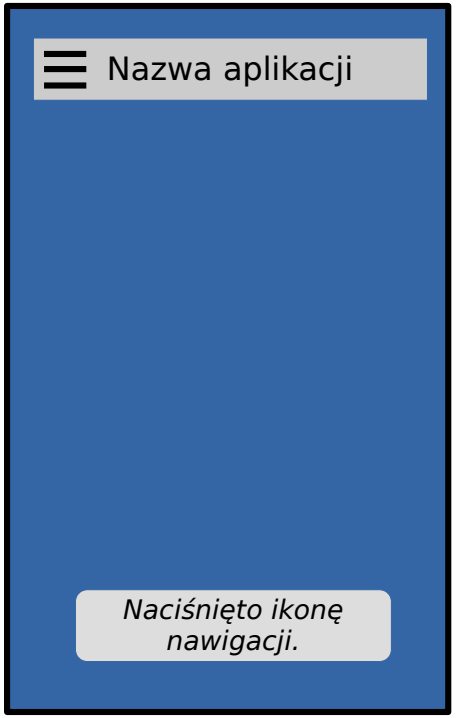
Hello Kitty!

Pasek tytułu (ver. 1)

```
@Composable
fun pasekTytulu() {
    Scaffold(
        topBar = {
            TopAppBar(
                colors = TopAppBarDefaults.smallTopAppBarColors(
                    containerColor = MaterialTheme.colorScheme.primaryContainer,
                    titleContentColor = MaterialTheme.colorScheme.primary
                ),
                title = {
                    Text("Tytuł aplikacji")
                }
            )
        }
    ){ // Wszystko, co ma być pod paskiem lub zostawić puste... }
}
```

Pasek tytułu (ver. 2)

```
val contextForToast = LocalContext.current.applicationContext
Column( // Tutaj jakieś opcje lub puste... ) {
 TopAppBar(
        title = {Text("Nazwa aplikacji")},
        navigationIcon = {IconButton(onClick = {
            Toast.makeText(contextForToast, "Naciśnięto ikonę nawigacji.",
            Toast.LENGTH_SHORT).show()
         }) {
            Icon(imageVector = Icons.Default.Menu,
            contentDescription = "Menu")
            }
        },
        colors = TopAppBarDefaults.topAppBarColors(
            containerColor = Color.LightGray,
            titleContentColor = Color.Magenta)
  )
}
```



Przycisk Chip

```
// Na zmianę: potwierdza kliknięcie / przybiera wartość domyślną
@Composable
fun przyciskFilterChip() {
    var selected by remember { mutableStateOf(false) }

    FilterChip(
        onClick = { selected = !selected },
        label = { Text("Kliknij mnie...") },
        selected = selected,
        leadingIcon = if (selected) {
            {
                Icon(
                    imageVector = Icons.Filled.Done,
                    contentDescription = "Ikona potwierdzenia",
                    modifier = Modifier.size(FilterChipDefaults.IconSize)
                )
            }
        } else {
            null
        }
    )
}
```

Sprawdzian

Wyświetla Toast’a z potwierdzeniem kliknięcia

Divider

Row otaczający Text i Box

Text

TextField

Image

Card

Button + Toast z potwierdzeniem kliknięcia