Android 02 Lista zakupów

Tworzymy nowy projekt.
Umieszczamy kod odpowiadający za tworzenie i modyfikowanie komponentów w oddzielnej funkcji kompozycyjnej:

```
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(null)


    //enableEdgeToEdge()
    setContent {
      ShoppingListTheme {
        Surface(modifier = Modifier.fillMaxSize()) {
          ComposableShoppingList()
            }
          }
        }
      }
}
```

Aplikacja powinna umożliwić tworzenie listy produktów do zakupienia w sklepie.
Z listy powinniśmy mieć możliwość usunięcia produktu oraz oznaczenia produktu jako zakupionego.
Dodatkowo możemy podać ilość produktu.

Klasa reprezentująca taki produkt może wyglądać następująco:

```
data class Product(
  val id:String,
  val name: String,
  val quantity: Int,
  var isPurchuased: Boolean = false
)
```

Wróćmy do metody **ComposableShoppingList()**
Najbardziej funkcjonalną metodą na dodanie produktu do listy od strony użytkownika wydaje się okno dialogowe pokazujące się po naciśnięciu przycisku.

```
@Composable
fun ComposableShoppingList(){}
```

Dodamy kolumnę a następnie przycisk:

```
Column(modifier = Modifier.fillMaxSize(),
  verticalArrangement = Arrangement.Center
) {
  Button(
    onClick = { showDialog = true },
    modifier = Modifier.align(Alignment.CenterHorizontally)
  )
  {
    Text("add product")
  }
  //...cd
  }
```

W metodzie onClick użyto zmiennej showDialog przypisując jej true.

Zmiany showDialog na true/false powinny pokazywać okno dialogowe lub je ukrywać,
należy utworzyć w naszej funkcji kompozycyjnej odpowiednią zmienną:

```
var showDialog by remember{ mutableStateOf(false) }
```

W funkcji kompozycyjnej powinien znajdować się warunek, który będzie decydował o tym czy pokazać dialog:

```
if (showDialog){ //...zawartość dialogu}
```

```
if (showDialog){
   AlertDialog(onDismissRequest = {showDialog=false},
      confirmButton = { Button(onClick = {}){ Text("ADD") }
      },
      title={Text("Add product to list")},
      text = {
         Column{
       TextField(value=productName, onValueChange = {productName=it})
              TextField(value=productQuantity, onValueChange = {productQuantity=it})
        }
      }
   )
}
```

AlertDialog zawiera onDismissRequest – kliknięcie poza obszar dialogu,
confirmButton – przycisk potwierdzenia,
title – tytuł,
text – obszar w którym umieścimy zawartość okna dialogowego
(w przykładzie kolumna z dwoma TextField)
Musimy utworzyć zmienne do manipulowania nazwą produktu i ilością podobnie jak showDialog:

```
var productName by remember{ mutableStateOf("") }
var productQuantity by remember{ mutableStateOf("") }
```

Dialog można zatwierdzić za pomocą confirmButton. Zostanie użyta do tego funkcja kompozycyjna Button. Należy w niej zaimplementować utworzenie nowego obiektu Product i dodanie go do listy.

Lista:

```
var  productList by rememberSaveable { mutableStateOf(listOf<Product>()) }
```

W przycisku potwierdzenia w metodzie onClick przekazujemy:

```
if(productName.isNotBlank()){
   val newProduct = Product(
      id = UUID.randomUUID().toString(),
      productName,
      productQuantity.toIntOrNull() ?: 1
   )
   productList = productList + newProduct
   showDialog = false
   productName=""
   productQuantity=""
}
```

Utworzono nowy obiekt Produkt i wstawiono do listy(zmiana referencji poprzez utworzenie nowej listy z dodatkowym produktem – zmiana referencji powoduje aktualizację UI przez jetpack compose). UUID tworzy nowe id.
Po dodaniu produktu należy przestać wyświetlać dialog oraz wyczyścić kontrolki do wprowadzania tekstu.

Wyświetlanie zawartości listy.
Lista będzie wyświetlana w LazyColumn. Elementy w tej kolumnie renderowane w miarę potrzeby. items(productList) będzie tworzyła dla każdego elementu z listy nową kompozycję (Composable) ProductRow. ProductRow jest kolejną funkcją kompozycyjną, którą należy stworzyć – będzie obsługiwała wyświetlanie elementów z listy na ekranie. Jest to funkcja „customowa” i należy ją napisać.

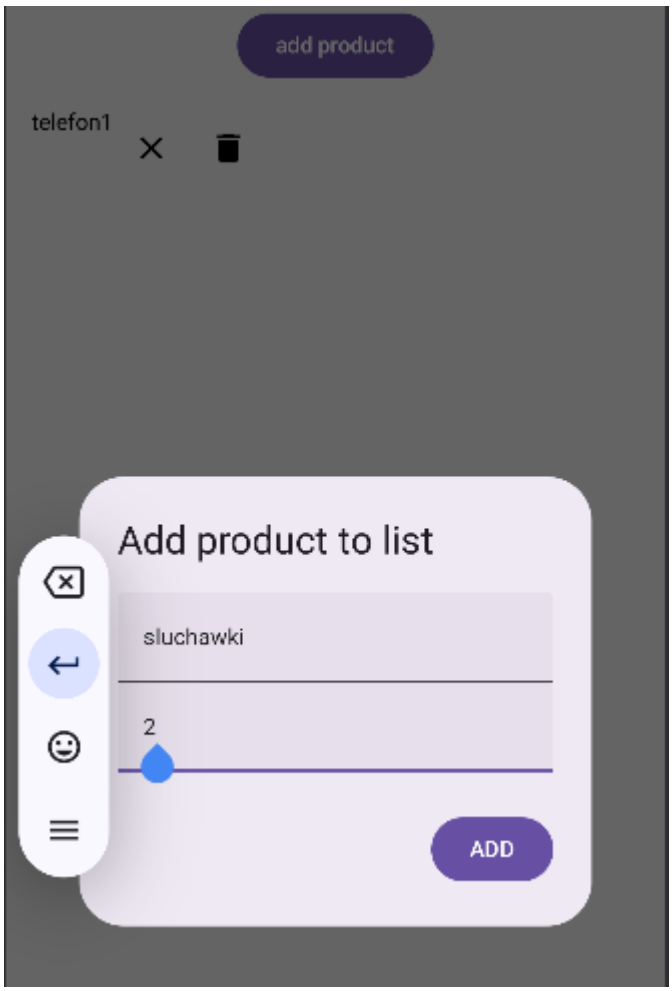
```
LazyColumn(modifier = Modifier
    .fillMaxSize()
    .padding(16.dp)) {
    items(productList) { product ->
        ProductRow(
            product = product,
            onPurhaseClick = {
                // Zmiana stanu produktu, tworząc nową listę z zaktualizowanym produktem
            },
            onDeleteClick = {
                // Usunięcie produktu z listy

            }
        )

    }
}
```

```
@Composable
fun ProductRow(
    product: Product,
    onPurhaseClick: () -> Unit,
    onDeleteClick: () -> Unit
) {
    Row( modifier=Modifier.fillMaxWidth()){
        //Text(text = product.id.toString())
        Text(text=product.name)
        Text(text=product.quantity.toString())
        IconButton(onClick = onPurhaseClick) {
            Icon(imageVector = if (product.isPurchuased==true) Icons.Default.Done else Icons.Default.Clear,
                "purchase icon")
        }
        IconButton(onClick = onDeleteClick) {
            Icon(imageVector = Icons.Default.Delete,"delete icon")
        }
    }
}
```

W nagłówku tej funkcji znajduje się produkt, oraz dwie funkcje, które należy przekazać.
Funkcja określająca zmianę statusu produktu – czy kupiono czy nie, oraz kliknięcie na przycisk usuwający produkt z listy.
Dodatkowo w wierszu znajdują się przyciski z ikonami do usuwania i zmiany statusu produktu.
W zależności od statusu produktu wyświetli się inna ikona dla przycisku.
Obsługa przycisków zostaje obsłużona za pomocą funkcji przekazanych do ProductRow (onPurchaseClick i onDeleteClick)



Aby móc zmieniać status produktu oraz umożliwić usuwanie z listy, należy zaimplementować funkcje przekazywane do ProductRow.

```
onPurhaseClick = {
    productList = productList.map {
        if (it.id == product.id) {
            it.copy(isPurchuased= !it.isPurchuased)
        } else {
            it
        }
    }
}
```

Funkcja **map** zwraca nową listę, w której każdy element jest przekształcony zgodnie z regułami określonymi w lambdzie, która jest do niej przekazana.

Dla elementu z listy, który ma id takie samo jak wybrane przez użytkownika(kliknięcie w przycisk z ikonką w danym wierszu produktu) tworzona jest nowa kopia produktu z odwróconą wartością pola isPurchased, reszta elementów zostaje taka sama.

Usuwanie również wymaga użycia metody zwracającej nową listę, przefiltrowaną po id produktu:

```
onDeleteClick = {
    productList = productList.filter { it.id != product.id }
}
```

Aby zachować listę w trakcie działania aplikacji np. podczas rotacji można wykorzystać

```
rememberSaveable
```

zamiast remeber.

```
var productList by rememberSaveable { mutableStateOf(listOf<Product>()) }
```

rememberSaveable przechowuje stan podczas renderowania Composable oraz przetrwa zmiany konfiguracji, takie jak rotacja ekranu.
Przy zamknięciu aplikacji należy jedna przechować w jakiś sposób listę produktów.

Cyklu Życia przy Obracaniu Ekranu:
Stara instancja aktywności:
onPause() → onStop() → onDestroy()
Nowa instancja aktywności:
onCreate() → onStart() → onResume()

Aby zapisać trwale listę prduktów należy skorzystać np. z SharedPreferenes, oraz narzędzia, które w wygony sposób zapisze do tekstu listę (w przykładzie będzie to json i biblioteką Gson).
SharedPreferences umożliwia przechowywanie prostych danych w postaci klucz-wartość w pamięci Androida.
context dla getSharedPreferences:
import androidx.compose.ui.platform.*LocalContext*
//context ma być Composable:

```
val context = LocalContext.current
val sharedPreferences = context.getSharedPreferences("list_prefs", Context.MODE_PRIVATE)
val gson = remember { Gson() }
```

Lista zostaje na początku wczytana z pliku:

```
var productList by rememberSaveable {
    mutableStateOf(loadProductList(sharedPreferences, gson))
}
```

Funkcja wczytująca listę produktów:

```
private fun loadProductList(sharedPreferences: android.content.SharedPreferences, gson: Gson): List<Product> {
    val jsonString = sharedPreferences.getString("product_list", null)
    return if (jsonString != null) {
        val type = object : TypeToken<List<Product>>() {}.type
        gson.fromJson(jsonString, type)
    } else {
        emptyList()
    }
}
```

pobieramy String z klucza product_list, jak brak klucza dostajemy null.
Jeżeli jest String gson przekształca stringa na List<Product>. Jeżeli json jest nullem zwraca pustą listę.

```
 object : TypeToken<List<Product>>() {}.type
```

zwraca typ na jaki gson ma przekształcić string. W JVM podczas kompilacji typy generyczne są wymazywane. List<Product> to List. Tworzymy nową klasę (anonimową), która przechowuje wszystkie informacje o typie generycznym.

Funkcja zapisująca listę produktów za pomocą gson:

```
private fun saveProductList(sharedPreferences: android.content.SharedPreferences, gson: Gson, productList:
List<Product>) {
    val editor = sharedPreferences.edit()
    val jsonString = gson.toJson(productList)
    editor.putString("product_list", jsonString)
    editor.apply()
}
```

Zapisanie produktów powinno się odbywać w odpowiednim przypadku – np. gdy zamykamy aplikację, aplikacja nie jest widoczna dla użytkownika:

```
val lifecycleOwner = LocalLifecycleOwner.current

DisposableEffect(lifecycleOwner) {
    val observer = LifecycleEventObserver { _, event ->
        if (event == Lifecycle.Event.ON_PAUSE) {
            Log.d("LifecycleOwnerCheck", "Current LifecycleOwner is: ${lifecycleOwner::class.simpleName}")

            saveProductList(sharedPreferences, gson, productList)
        }
    }

    lifecycleOwner.lifecycle.addObserver(observer)

    onDispose {
        lifecycleOwner.lifecycle.removeObserver(observer)
    }
}
```

DisposableEffect zostaje uruchomione przy wyrenderowaniu kompozycji.
Tworzony jest obserwator cyklu życia aktywności (właścicielem jest aktywność), gdy cykl życia ma status ON_PAUSE zapisujemy listę,
onDispose jest wywoływane, kiedy kompozycja przestaje być aktywną częścią UI -usuwany jest obserwator.
W logach można sprawdzić kiedy jest tworzony i usuwany:

```
Log.d("ADD OBSERVER", "Owner: ${lifecycleOwner::class.simpleName}")
```

```
Log.d("REMOVE OBSERVER", "Owner: ${lifecycleOwner::class.simpleName}")
```

Zadanie2:
Stwórz listę zakupów. Lista powinna zawierać nazwy produktów (wpisywane dowolnie), ilość/liczbę opakowań oraz oznaczenie, czy dany produkt został kupiony. Aplikacja musi mieć możliwość odtworzenia zapisanej listy pomimo wyjścia z aplikacji. Aplikacja musi mieć możliwość usuwania produktu z listy.

Importy i zależności:

```
import android.content.Context

import android.os.Bundle
import android.util.Log
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row

import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth

import androidx.compose.foundation.layout.padding

import androidx.compose.material3.Button
import androidx.compose.material3.Surface
import androidx.compose.material3.Text
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Clear
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Done
import androidx.compose.material3.AlertDialog

import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
import androidx.compose.runtime.DisposableEffect
import androidx.compose.runtime.saveable.rememberSaveable
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.platform.LocalLifecycleOwner
import androidx.compose.ui.tooling.preview.Preview
import androidx.compose.ui.unit.dp
import androidx.lifecycle.Lifecycle
import androidx.lifecycle.LifecycleEventObserver
```

```
import com.example.shoppinglist.ui.theme.ShoppingListTheme
import com.google.gson.Gson
import com.google.gson.reflect.TypeToken
import java.util.UUID
```

```
ependencies {


    implementation(libs.androidx.core.ktx)
    implementation(libs.gson)
    implementation(libs.androidx.lifecycle.runtime.ktx)
    implementation(libs.androidx.activity.compose)
    implementation(platform(libs.androidx.compose.bom))
    implementation(libs.androidx.ui)
    implementation(libs.androidx.ui.graphics)
    implementation(libs.androidx.ui.tooling.preview)
    implementation(libs.androidx.material3)
    testImplementation(libs.junit)
    androidTestImplementation(libs.androidx.junit)
    androidTestImplementation(libs.androidx.espresso.core)
    androidTestImplementation(platform(libs.androidx.compose.bom))
    androidTestImplementation(libs.androidx.ui.test.junit4)
    debugImplementation(libs.androidx.ui.tooling)
    debugImplementation(libs.androidx.ui.test.manifest)
}
```

Jeżeli robimy nowe importowanie z Version Catalog
`implementation(libs.gson)`
trzeba dodać gson w libs.versions.toml:

[versions]
agp = "8.6.0"
gson = "2.11.0"
kotlin = "1.9.0"
coreKtx = "1.13.1"
junit = "4.13.2"
junitVersion = "1.2.1"
espressoCore = "3.6.1"
lifecycleRuntimeKtx = "2.8.6"
activityCompose = "1.9.2"
composeBom = "2024.04.01"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
gson = {group= "com.google.code.gson", name="gson", version.ref = "gson" }
junit = { group = "junit", name = "junit", version.ref = "junit" }
androidx-junit = { group = "androidx.test.ext", name = "junit", version.ref = "junitVersion" }
androidx-espresso-core = { group = "androidx.test.espresso", name = "espresso-core", version.ref =

"espressoCore" }
androidx-lifecycle-runtime-ktx = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version.ref = "lifecycleRuntimeKtx" }
androidx-activity-compose = { group = "androidx.activity", name = "activity-compose", version.ref = "activityCompose" }
androidx-compose-bom = { group = "androidx.compose", name = "compose-bom", version.ref = "composeBom" }
androidx-ui = { group = "androidx.compose.ui", name = "ui" }
androidx-ui-graphics = { group = "androidx.compose.ui", name = "ui-graphics" }
androidx-ui-tooling = { group = "androidx.compose.ui", name = "ui-tooling" }
androidx-ui-tooling-preview = { group = "androidx.compose.ui", name = "ui-tooling-preview" }
androidx-ui-test-manifest = { group = "androidx.compose.ui", name = "ui-test-manifest" }
androidx-ui-test-junit4 = { group = "androidx.compose.ui", name = "ui-test-junit4" }
androidx-material3 = { group = "androidx.compose.material3", name = "material3" }

[plugins]
android-application = { id = "com.android.application", version.ref = "agp" }
kotlin-android = { id = "org.jetbrains.kotlin.android", version.ref = "kotlin" }