|  |  |
| --- | --- |
| Logo AMW Gdynia | **AKADEMIA MARYNARKI WOJENNEJ**  **im. BOHATERÓW WESTERPLATTE**  **WYDZIAŁ MECHANICZNO-ELEKTRYCZNY**  **KATEDRA INFORMATYKI** |

**SPRAWOZDANIE**

**Sortowanie liczb   
wykorzystując fragmenty i aktywności**

**Android Studio**

|  |  |
| --- | --- |
|  | Wykonawca: |
|  | **Patrycja Przybysz** |
|  | Grupa: |
|  | **215IC B2** |

GDYNIA 2024

1. Projekt i architektura aplikacji z jedną aktywnością i fragmentami

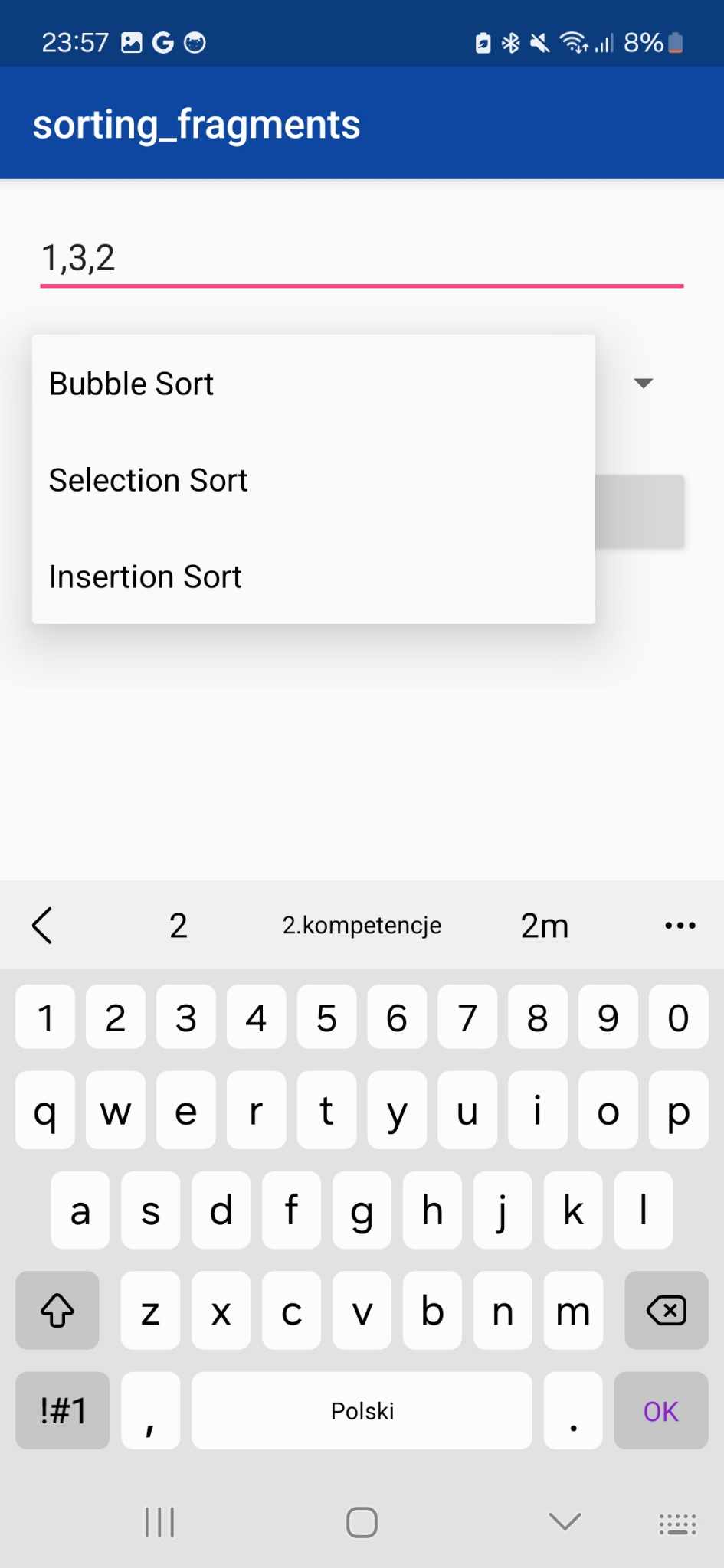
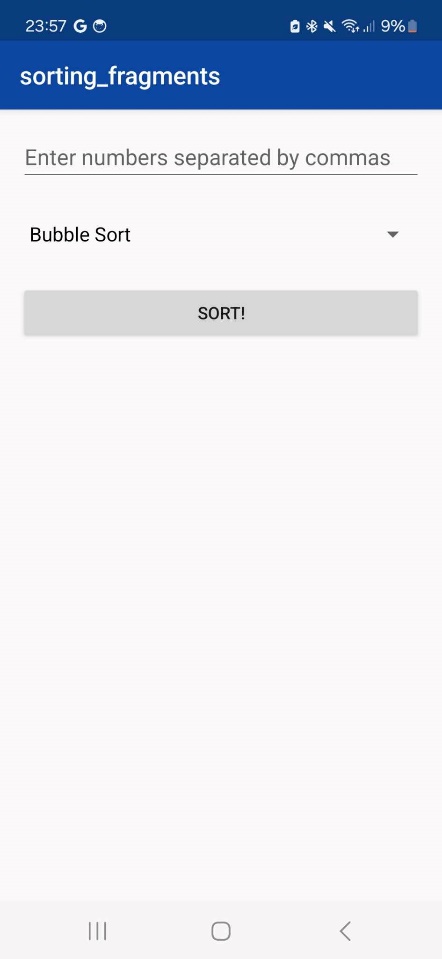
Aplikacja została zaimplementowana w architekturze Model-View-ViewModel (MVVM). Oznacza to, że warstwa prezentacji jest oddzielona od logiki.

Struktura projektu obejmuje:

1. Katalog layout zawierający pliki XML dla interfejsu użytkownika. Należą do nich:

* fragment\_sorting\_method.xml
* fragment\_sorting\_execution.xml
* main.xml

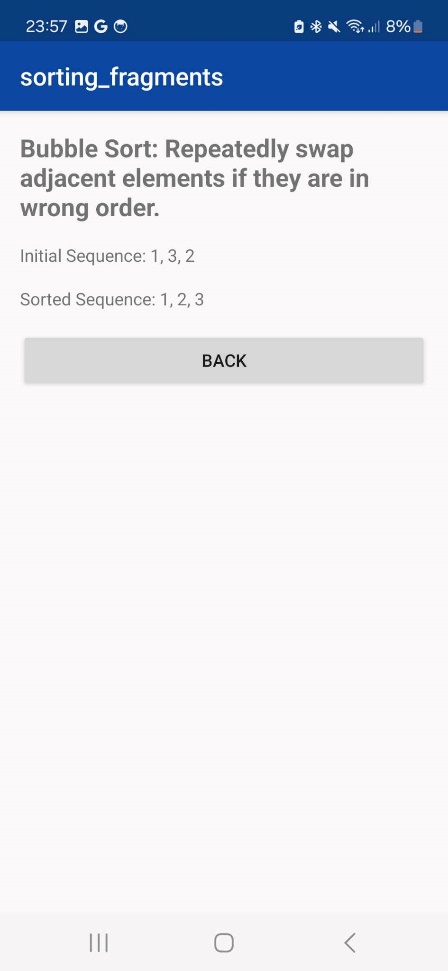
Plik **fragment\_sorting\_method.xml** zawiera wygląd aplikacji tuż po jej odpaleniu tzn. miejsce do wprowadzenia liczb przez użytkownika (TextView), listę z wyborem metody do sortowania (Spinner), oraz przycisk do wywoływania drugiego fragmentu (Button).



Rysunek 1 Wygląd aplikacji

Rysunek 2 Wygląd aplikacji

Plik **fragment\_sorting\_execution.xml** zawiera wygląd aplikacji po naciśnięciu przycisku „SORT!”. Zawiera on pole z krótkim opisem wybranej metody sortowania (TextView), pole z nieposortowanym ciągiem liczb i z posortowanym ciągiem oraz przycisk (Button) „Back” powodujący powrót do strony głównej (pierwszy fragment).



Rysunek 3 Wygląd aplikacji

Plik **main.xml** to plik który zawiera definicję widoku FragmentContainerView, który używany jest w układzie (layout) interfejsu użytkownika do wyświetlania fragmentów w aplikacji.

<?xml version="1.0" encoding="utf-8"?>  
<androidx.fragment.app.FragmentContainerView xmlns:android="http://schemas.android.com/apk/res/android"  
 android:id="@+id/fragment\_container"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />

1. Katalog java zawierający klasy Kotlin, w tym:

* **MainActivity** - główna aktywność, odpowiadająca za nawigację między fragmentami i przetwarzanie danych.
* **SortingExecutionFragment** - fragment odpowiedzialny za wykonywanie sortowania i prezentację wyników.
* **SortingMethodFragment** - fragment umożliwiający użytkownikowi wybór metody sortowania oraz pozwalający na wprowadzenie liczb

1. Implementacja aplikacji z aktywnością i dwoma fragmentami
2. MainActivity:

* Główna aktywność aplikacji, która służy jako kontener dla fragmentów.
* Implementuje metodę onCreate, która ustawia układ interfejsu użytkownika, w tym kontener na fragmenty.
* Zarządza przełączaniem między fragmentami, reagując na akcje użytkownika, takie jak wybór metody sortowania i przechodzenie do fragmentu sortowania.

package com.example.sorting\_fragments  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
  
class MainActivity : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
  
 // Ustawienie układu interfejsu użytkownika  
 setContentView(R.layout.main)  
  
 // Sprawdzenie czy istnieje zapisany stan aktywności  
 if (savedInstanceState == null) {  
 // Dodanie fragmentu SortingMethodFragment do kontenera fragmentów  
 *supportFragmentManager*.beginTransaction()  
 .replace(R.id.*fragment\_container*, SortingMethodFragment())  
 .commitNow()  
 }  
 }  
}

1. SortingMethodFragment:

* Fragment odpowiedzialny za wyświetlanie listy metod sortowania i obsługę wyboru metody przez użytkownika.
* Zawiera listę dostępnych metod sortowania (np. Bubble Sort, Selection Sort, Insertion Sort) w formie listy rozwijanej
* W metodzie onCreateView fragmentu tworzony jest interfejs użytkownika za pomocą pliku layoutu fragment\_sorting\_method.xml.
* Po kliknięciu przycisku „SORT!”, pobierane są wprowadzone przez użytkownika liczby i wybrana metoda sortowania.
* Dane te są przekazywane do nowego obiektu Bundle, który jest przekazywany jako argument do fragmentu SortingExecutionFragment.

class SortingMethodFragment : Fragment() {  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
   
 // Nadaje widokowi dla tego fragmentu zawartość z pliku layoutu  
 val view = inflater.inflate(R.layout.*fragment\_sorting\_method*, container, false)  
  
   
 // Inicjalizacja komponentów interfejsu użytkownika  
 val inputNumbers = view.findViewById<EditText>(R.id.*inputNumbers*)  
 val sortMethodSpinner = view.findViewById<Spinner>(R.id.*sortMethodSpinner*)  
 val nextButton = view.findViewById<Button>(R.id.*nextButton*)  
  
   
 // Wypełnienie spinnera dostępnymi metodami sortowania  
 val sortMethods = *arrayOf*("Bubble Sort", "Selection Sort", "Insertion Sort")  
 sortMethodSpinner.*adapter* = ArrayAdapter(  
 requireContext(),  
 android.R.layout.*simple\_spinner\_dropdown\_item*,  
 sortMethods  
 )  
  
   
 // Obsługa kliknięcia przycisku "Next"  
 nextButton.setOnClickListener **{** // Pobranie wprowadzonych liczb oraz wybranej metody sortowania  
 val numbers = inputNumbers.*text*.toString()  
 val selectedMethod = sortMethodSpinner.*selectedItem*.toString()  
  
   
 // Utworzenie pakietu danych do przekazania do SortingExecutionFragment  
 val bundle = Bundle().*apply* **{** putString("numbers", numbers)  
 putString("method", selectedMethod)  
 **}** // Utworzenie fragmentu SortingExecutionFragment i przekazanie pakietu jako argumentów  
 val sortingExecutionFragment = SortingExecutionFragment().*apply* **{** *arguments* = bundle  
 **}** // Zastąpienie bieżącego fragmentu fragmentem SortingExecutionFragment  
 requireActivity().*supportFragmentManager*.beginTransaction()  
 .replace(R.id.*fragment\_container*, sortingExecutionFragment)  
 .addToBackStack(null)  
 .commit()  
 **}**return view**}}**

1. SortingExecutionFragment:

* Fragment odpowiedzialny za interakcję z użytkownikiem, pobranie listy liczb i przeprowadzenie sortowania z wybranej metody.
* Pobiera dane wejściowe (liczby i metoda sortowania) z argumentów przekazanych przez poprzedni fragment.
* Wyświetla wyniki sortowania w interfejsie użytkownika, prezentując zarówno początkową listę liczb, jak i posortowaną w wybrany sposób.
* zawiera implementacje 3 metod sortowania (Bubble sort, insertion sort i selection sort)
* Obsługuje również kliknięcie przycisku "Back", które powoduje nawigację z powrotem do poprzedniego fragmentu.
* class SortingExecutionFragment : Fragment() {  
    
   override fun onCreateView(  
   inflater: LayoutInflater, container: ViewGroup?,  
   savedInstanceState: Bundle?  
   ): View? {  
     
   // Nadaje widokowi dla tego fragmentu zawartość z pliku layoutu  
   val view = inflater.inflate(R.layout.*fragment\_sorting\_execution*, container, false)  
    
     
   // Inicjalizacja komponentów interfejsu użytkownika  
   val sortMethodDescription = view.findViewById<TextView>(R.id.*sortMethodDescription*)  
   val initialSequence = view.findViewById<TextView>(R.id.*initialSequence*)  
   val sortedSequence = view.findViewById<TextView>(R.id.*sortedSequence*)  
   val backButton = view.findViewById<Button>(R.id.*backButton*)  
    
     
   // Pobranie danych wejściowych z pakietu argumentów  
   val numbers = *arguments*?.getString("numbers")!!.*split*(",").*map* **{ it**.*trim*().*toInt*() **}**.*toMutableList*()  
   val originalNumbers = numbers.*toList*()  
   val method = *arguments*?.getString("method")!!  
    
     
   // Wykonanie sortowania na podstawie wybranej metody  
   val sortedNumbers = when (method) {  
   "Bubble Sort" -> bubbleSort(numbers)  
   "Selection Sort" -> selectionSort(numbers)  
   "Insertion Sort" -> insertionSort(numbers)  
   else -> numbers  
   }  
    
     
   // Ustawienie opisu, początkowej sekwencji i posortowanej sekwencji  
   val description = when (method) {  
   "Bubble Sort" -> "Bubble Sort: Repeatedly swap adjacent elements if they are in wrong order."  
   "Selection Sort" -> "Selection Sort: Select the smallest element from an unsorted list and swap it with the leftmost unsorted element."  
   "Insertion Sort" -> "Insertion Sort: Insert each element into its proper place in the sorted part of the list."  
   else -> ""  
   }  
   sortMethodDescription.*text* = description  
   initialSequence.*text* = "Initial Sequence: ${originalNumbers.*joinToString*(", ")}"  
   sortedSequence.*text* = "Sorted Sequence: ${sortedNumbers.*joinToString*(", ")}"  
    
     
   // Obsługa kliknięcia przycisku powrotu do nawigacji z powrotem do SortingMethodFragment  
   backButton.setOnClickListener **{** requireActivity().*supportFragmentManager*.popBackStack()  
   **}** return view  
   }  
    
     
   // Algorytm sortowania bąbelkowego  
   private fun bubbleSort(list: MutableList<Int>): List<Int> {  
   for (i in list.*indices*) {  
   for (j in 0 *until* list.size - i - 1) {  
   if (list[j] > list[j + 1]) {  
   val temp = list[j]  
   list[j] = list[j + 1]  
   list[j + 1] = temp  
   }  
   }  
   }  
   return list  
   }  
    
     
   // Algorytm sortowania przez wybieranie  
   private fun selectionSort(list: MutableList<Int>): List<Int> {  
   for (i in list.*indices*) {  
   var minIdx = i  
   for (j in i + 1 *until* list.size) {  
   if (list[j] < list[minIdx]) {  
   minIdx = j  
   }  
   }  
   val temp = list[i]  
   list[i] = list[minIdx]  
   list[minIdx] = temp  
   }  
   return list  
   }  
    
     
   // Algorytm sortowania przez wstawianie  
   private fun insertionSort(list: MutableList<Int>): List<Int> {  
   for (i in 1 *until* list.size) {  
   val key = list[i]  
   var j = i - 1  
   while (j >= 0 && list[j] > key) {  
   list[j + 1] = list[j]  
   j -= 1  
   }  
   list[j + 1] = key  
   }  
   return list  
   }  
  }

1. Projekt i architektura aplikacji z dwiema aktywnościami

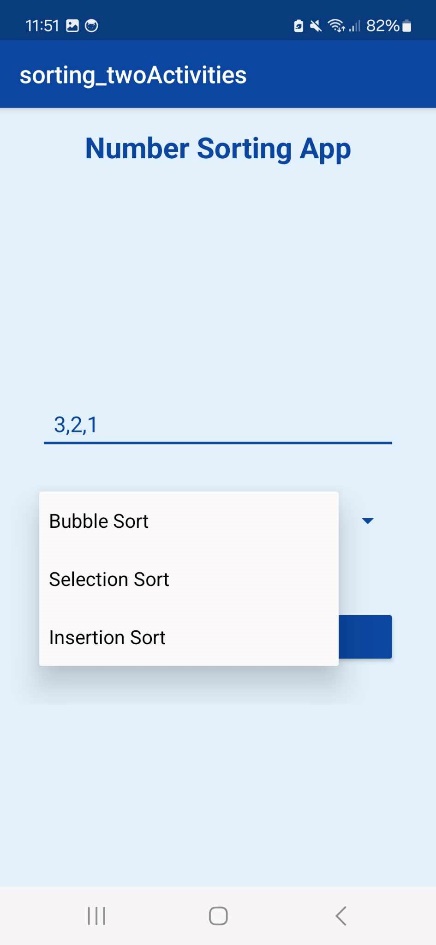
Aplikacja została zaimplementowana w architekturze Model-View-ViewModel (MVVM). Oznacza to, że warstwa prezentacji jest oddzielona od logiki.

Struktura projektu obejmuje:

1. Katalog layout zawierający pliki XML dla interfejsu użytkownika. Należą do nich:

* activity\_main.xml
* activity\_sorting.xml

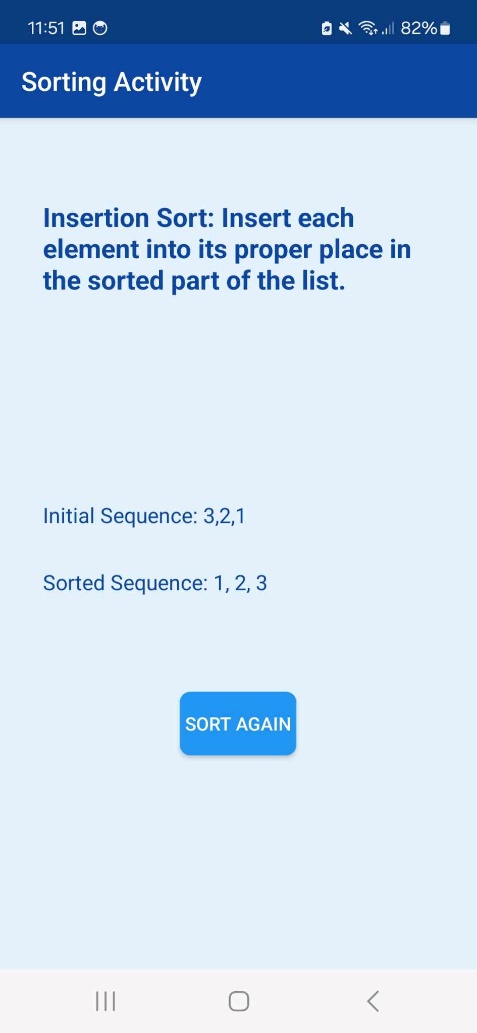
Plik **activity\_main.xml** zawiera definicję elementów i ich wygląd tuż po uruchomieniu aplikacji. Zawiera miejsce do wprowadzenia liczb przez użytkownika (TextView), listę z wyborem metody do sortowania (Spinner), oraz przycisk do wywoływania drugiej aktywności (Button).



Rysunek 4 Wygląd aplikacji

Rysunek 5 Wygląd aplikacji

Plik **activity\_sorting.xml** zawiera wygląd drugiej aktywności tzn. TextView z opisem wybranej metody, wyświetlony nieposortowany ciąg liczb i posortowany ciag, oraz przycisk powrotu do poprzedniej aktywności „SORT AGAIN”.



Rysunek 6 Wygląd aplikacji

1. Katalog drawable zawierający pliki xml, które zmieniają wygląd tła, przycisków, pól edytowalnych. Należą do nich:

* button\_background.xml
* edit\_text\_background.xml
* oraz pliki ic\_launcher.xml zmieniające ikonkę aplikacji

1. Katalog java zawierający klasy Kotlin:

* MainActivity.kt
* SortingActivity.kt

1. Plik AndroidManifest.xml w którym umieszcza się aktywności
2. Implementacja
3. MainActivity:

* Metoda onCreate jest wywoływana podczas tworzenia aktywności. I ustawiany jest layout activity\_main
* Utworzenie tablicy sortMethods zawierającej nazwy metod sortowania.
* Pobranie wprowadzonych liczb jako ciąg tekstowy i metody sortowania ze spinnera
* Utworzenie intencji Intent, aby przejść do SortingActivity.
* Dołączenie danych (liczby i metoda sortowania) do intencji.

class MainActivity : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*) // Ustawienie layoutu activity\_main  
  
 val inputNumbers = findViewById<EditText>(R.id.*inputNumbers*) // Inicjalizacja EditText dla wprowadzania liczb  
 val sortMethodSpinner = findViewById<Spinner>(R.id.*sortMethodSpinner*) // Inicjalizacja Spinner dla wyboru metody sortowania  
 val nextButton = findViewById<Button>(R.id.*nextButton*) // Inicjalizacja Button dla przycisku "Next"  
  
 val sortMethods = *arrayOf*("Bubble Sort", "Selection Sort", "Insertion Sort") // Definicja metod sortowania  
 sortMethodSpinner.*adapter* = ArrayAdapter(this, android.R.layout.*simple\_spinner\_dropdown\_item*, sortMethods) // Ustawienie adaptera dla Spinner  
  
 nextButton.setOnClickListener **{** val numbers = inputNumbers.*text*.toString() // Pobranie wprowadzonych liczb jako ciąg tekstowy  
 val selectedMethod = sortMethodSpinner.*selectedItem*.toString() // Pobranie wybranej metody sortowania z Spinner  
  
 val intent = Intent(this, SortingActivity::class.*java*).*apply* **{** putExtra("numbers", numbers) // Dodanie liczb do intencji  
 putExtra("method", selectedMethod) // Dodanie wybranej metody sortowania do intencji  
 **}** startActivity(intent) // Rozpoczęcie nowej aktywności SortingActivity  
 **}** }  
}

1. SortingActivity:

* Metoda onCreate jest wywoływana podczas tworzenia aktywności oraz ustawiany jest layout activity\_sorting.
* Pobranie wprowadzonych liczb z intencji jako ciąg tekstowy i podzielenie go na listę liczb całkowitych oraz pobranie oryginalnej sekwencji liczb z intencji i metody sortowania
* Wykonanie sortowania na podstawie wybranej metody oraz pobranie opisu metody sortowania
* Po kliknięciu przycisku "Back" uruchamiana jest metoda goBack, która powoduje powrót do MainActivity.
* Implementacja metod sortowania Bubble sortt insert sort oraz selection sort

package com.example.sorting\_twoactivities  
  
import android.content.Intent  
import android.os.Bundle  
import android.widget.Button  
import android.widget.TextView  
import androidx.appcompat.app.AppCompatActivity  
  
class SortingActivity : AppCompatActivity() {  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_sorting*) // Ustawienie layoutu activity\_sorting  
  
 // Inicjalizacja TextView dla opisu metody sortowania, początkowej sekwencji i posortowanej sekwencji  
 val sortMethodDescription = findViewById<TextView>(R.id.*sortMethodDescription*)  
 val initialSequence = findViewById<TextView>(R.id.*initialSequence*)  
 val sortedSequence = findViewById<TextView>(R.id.*sortedSequence*)  
  
 // Pobranie danych z Intent: liczby, oryginalna sekwencja oraz wybrana metoda sortowania  
 val numbers = *intent*.getStringExtra("numbers")!!.*split*(",").*map* **{ it**.*trim*().*toInt*() **}**.*toMutableList*()  
 val originalNumbers = *intent*.getStringExtra("originalNumbers")  
 val method = *intent*.getStringExtra("method")!!  
  
 // Wykonanie sortowania na podstawie wybranej metody  
 val sortedNumbers = when (method) {  
 "Bubble Sort" -> bubbleSort(numbers)  
 "Selection Sort" -> selectionSort(numbers)  
 "Insertion Sort" -> insertionSort(numbers)  
 else -> numbers  
 }  
  
 // Pobranie opisu wybranej metody sortowania  
 val description = when (method) {  
 "Bubble Sort" -> "Bubble Sort: Repeatedly swap adjacent elements if they are in wrong order."  
 "Selection Sort" -> "Selection Sort: Select the smallest element from an unsorted list and swap it with the leftmost unsorted element."  
 "Insertion Sort" -> "Insertion Sort: Insert each element into its proper place in the sorted part of the list."  
 else -> ""  
 }  
  
 // Ustawienie tekstów dla opisów metody sortowania, początkowej sekwencji oraz posortowanej sekwencji  
 sortMethodDescription.*text* = description  
 initialSequence.*text* = "Initial Sequence: ${originalNumbers ?: numbers.*joinToString*(", ")}"  
 sortedSequence.*text* = "Sorted Sequence: ${sortedNumbers.*joinToString*(", ")}"  
  
 // Inicjalizacja przycisku "Back" i ustawienie jego akcji kliknięcia  
 val backButton = findViewById<Button>(R.id.*backButton*)  
 backButton.setOnClickListener **{** goBack() // Powrót do MainActivity  
 **}** }  
  
 // Metoda do powrotu do MainActivity  
 private fun goBack() {  
 val intent = Intent(this, MainActivity::class.*java*)  
 startActivity(intent)  
 }  
  
 // Implementacja metody sortowania bąbelkowego (Bubble Sort)  
 private fun bubbleSort(list: MutableList<Int>): List<Int> {  
 for (i in list.*indices*) {  
 for (j in 0 *until* list.size - i - 1) {  
 if (list[j] > list[j + 1]) {  
 val temp = list[j]  
 list[j] = list[j + 1]  
 list[j + 1] = temp  
 }  
 }  
 }  
 return list  
 }  
  
 // Implementacja metody sortowania przez wybór (Selection Sort)  
 private fun selectionSort(list: MutableList<Int>): List<Int> {  
 for (i in list.*indices*) {  
 var minIdx = i  
 for (j in i + 1 *until* list.size) {  
 if (list[j] < list[minIdx]) {  
 minIdx = j  
 }  
 }  
 val temp = list[i]  
 list[i] = list[minIdx]  
 list[minIdx] = temp  
 }  
 return list  
 }  
  
 // Implementacja metody sortowania przez wstawianie (Insertion Sort)  
 private fun insertionSort(list: MutableList<Int>): List<Int> {  
 for (i in 1 *until* list.size) {  
 val key = list[i]  
 var j = i - 1  
 while (j >= 0 && list[j] > key) {  
 list[j + 1] = list[j]  
 j -= 1  
 }  
 list[j + 1] = key  
 }  
 return list  
 }  
}