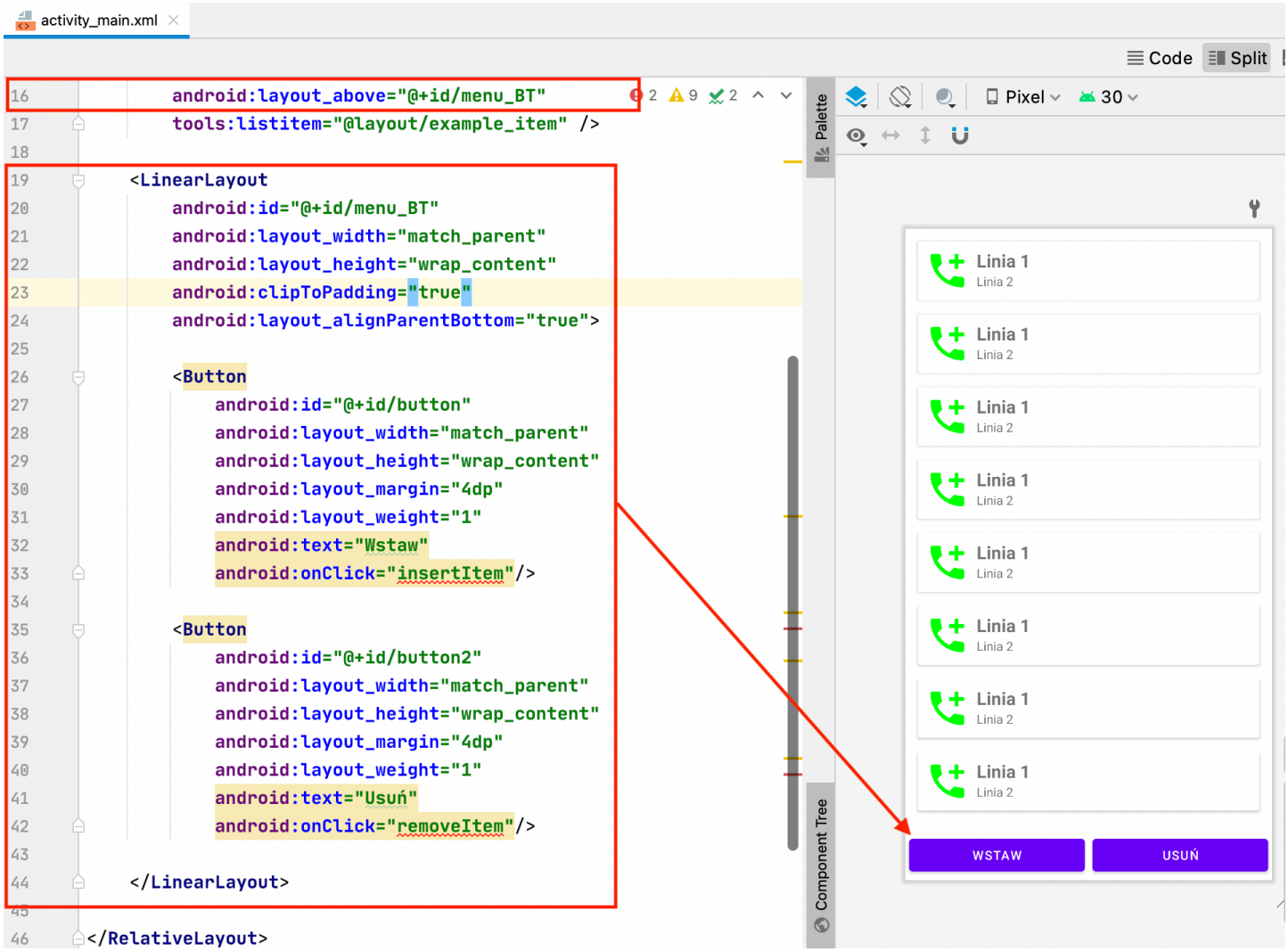
Obsługa RecyclerView – dodanie przycisków modyfikujących

Zajmiemy się kolejnym elementem jakim jest dodanie przycisków obsługujących poszczególne pozycje listy.

Punktem wyjściowym jest utworzona lista. W pliku activity_main.xml dodamy layout z dwoma przyciskami:




```
16          android:layout_above="@+id/menu_BT"
17          tools:listitem="@layout/example_item" />
18
19      <LinearLayout
20          android:id="@+id/menu_BT"
21          android:layout_width="match_parent"
22          android:layout_height="wrap_content"
23          android:clipToPadding="true"
24          android:layout_alignParentBottom="true">
25
26          <Button
27              android:id="@+id/button"
28              android:layout_width="match_parent"
29              android:layout_height="wrap_content"
30              android:layout_margin="4dp"
31              android:layout_weight="1"
32              android:text="Wstaw"
33              android:onClick="insertItem"/>
34
35          <Button
36              android:id="@+id/button2"
37              android:layout_width="match_parent"
38              android:layout_height="wrap_content"
39              android:layout_margin="4dp"
40              android:layout_weight="1"
41              android:text="Usuń"
42              android:onClick="removeItem"/>
43
44      </LinearLayout>
45
46  </RelativeLayout>
```

Teraz trzeba oprogramować zachowanie przycisków w pliku MainActivity.kt .

Dodajemy dwie funkcje obsługujące oba przyciski:

MainActivity.kt

```
3  import android.os.Bundle
4  import android.view.View
5  import androidx.appcompat.app.AppCompatActivity
6  import androidx.recyclerview.widget.LinearLayoutManager
7  import androidx.recyclerview.widget.RecyclerView
8  import com.example.recyclerviewapp.databinding.ActivityMainBinding
9  import kotlin.random.Random
10
11 class MainActivity : AppCompatActivity() {
12     lateinit var binding: ActivityMainBinding
13     private val exampleList = generuj( size: 500)
14     private val adapter = ExampleAdapter(exampleList)
15     override fun onCreate(savedInstanceState: Bundle?) {
16         super.onCreate(savedInstanceState)
17         //setContentView(R.layout.activity_main)
18         binding = ActivityMainBinding.inflate(layoutInflater)
19         setContentView(binding.root)
20         //val exampleList = generuj(500) //zrobić zmienną 'exampleList' jako globalną
21         binding.recyclerView.adapter = adapter //przerobiona zmienna 'adapter'
22         binding.recyclerView.layoutManager = LinearLayoutManager( context: this)
23         binding.recyclerView.setHasFixedSize(true)
24     }
25     fun insertItem(view:View){
26         val index = Random( seed: 8) //dodanie elementu o indeksie 8
27         val newItem = ExampleItem(
28             R.drawable.ic_baseline,
29             text1 = "Nowy element $index",
30             text2 = "Linia 2.$index")
31     }
32     fun removeItem(view:View){
33
34     }
```

Przerabiamy funkcję z listy na tablicę

```
36     private  fun generuj(size:Int):ArrayList<ExampleItem>{
37         val list = ArrayList<ExampleItem>()
```

Oraz zatwierdzamy zmiany w funkcji dodającej pozycję listy (tablicy). Nowa postać funkcji:

MainActivity.kt

```
25     fun insertItem(view:View){
26         val index = Random.nextInt( until: 8) //dodanie elementu o indeksie 8
27         val newItem = ExampleItem(
28             R.drawable.ic_baseline,
29             text1 = "Nowy element $index",
30             text2 = "Linia 2.$index")
31         exampleList.add(index,newItem)
32         adapter.notifyItemInserted(index)
33     }
```

Podobnie wprowadzamy zmiany do funkcji usuwającej dane:

MainActivity.kt

```
33      }
34      fun removeItem(view:View){
35          val index = Random.nextInt( until: 8)
36          exampleList.removeAt(index)
37          adapter.notifyItemRemoved(index)
38      }
```

Uzyskaliśmy możliwość dodawania i usuwania elementów listy w sposób losowy z zakresu 8 losowych elementów.

Zajmiemy się teraz klikaniem/zaznaczaniem poszczególnych pozycji na liście.

Zmienimy zachowanie się klasy odpowiedzialnej za widoczność elementów listy **ExampleViewHolder**. Rozszerzymy dziedziczenie klasy o metodę OnClickListener:

ExampleAdapter.kt | View.java

```
28514   /**
28515    * Interface definition for a callback to be invoked when a view is clicked.
28516    */
28517   public interface OnClickListener {
28518       /**
28519        * Called when a view has been clicked.
28520        *
28521        * @param v The view that was clicked.
28522        */
28523       void onClick(View v);
28524   }
```

Dopisujemy dziedziczenie

ExampleAdapter.kt

```
26      //klasa odpowiedzialna za widoczność elementów listy
27      class ExampleViewHolder(itemView:View):RecyclerView.ViewHolder(itemView),View{
28          val imageView:ImageView = itemView.findViewById(
29          val textView1:TextView = itemView.findViewById(R
```

View (android.view)
ViewGroup (android.view)

ExampleAdapter.kt

```
26      //klasa odpowiedzialna za widoczność elementów listy
27      class ExampleViewHolder(itemView:View):RecyclerView.ViewHolder(itemView),View.OnClic {
28          val imageView:ImageView = itemView.findViewById(R.id.image_vie
29          val textView1:TextView = itemView.findViewById(R.id.text_view_
```

OnClickListener (android.view.View)
OnContextClickListener (android.view

Inicjujemy zachowanie kliknięcia

```
ExampleAdapter.kt
26      //klasa odpowiedzialna za widoczność elementów listy
27      class ExampleViewHolder(itemView:View):RecyclerView.Vie
28          val imageView:ImageView = itemView.findViewById(R.i
29          val textView1:TextView = itemView.findViewById(R.id
30          val textView2:TextView = itemView.findViewById(R.id
31          init{
32              itemView.set
33          }
34      }
35  }
36
```

```
setBackgroundColor(color: Int)
background (from getBackground()/setBac
setOnClickListener(l: View.OnClickListe
setOnClickListener {...} (l: ((View!) -
```

```
31          init{
32              itemView.setOnClickListener(this)
33          }
```

Funkcja odpowiedzialna za klikanie:

```
31          init{
32              itemView.setOnClickListener(this)
33          }
34          override fun onC
35      }
36  }
37
```

```
override fun onClick(v: View?) {...}    OnClickListener
Press ⏎ to insert, ⇥ to replace
```

Utworzymy interfejs użytkownika, czyli interakcję programu na kliknięcie

```
ExampleAdapter.kt
37          init{
38              itemView.setOnClickListener(this)
39          }
40
41          override fun onClick(v: View?) {
42              TODO()
43          }
44      }
45      interface OnItemClickListener {
46          fun onItemClick(position: Int)
47      }
48  }
```

Dodajemy kolejny atrybut klasy adaptera. Atrybut będzie odpowiedzialny za kliknięcie w element listy

```
11  class ExampleAdapter(private  val exampleList: List<ExampleItem>
12      //tutaj dodajemy argument
13      ):RecyclerView.Adapter<ExampleAdapter.ExampleViewHolder>() {
14
```

```
12
13  class ExampleAdapter(
14      private  val exampleList: List<ExampleItem>,
15      private val listener: OnIt
16      ) :RecyclerView.Ada
```

OnItemClickListener (android.widget.AdapterView)

Po dopisaniu argumentu:

ExampleAdapter.kt

```
14  class ExampleAdapter(
15      private  val exampleList: List<ExampleItem>,
16      private val listener: OnItemClickListener
17      ) :RecyclerView.Adapter<ExampleAdapter.ExampleViewHolder>() {
```

Przerabiamy teraz klasę ExampleViewHolder na klasę wewnętrzną oraz przerabiamy zachowanie funkcji w klasie:

```
31      //klasa odpowiedzialna za widoczność elementów tls
32      inner class ExampleViewHolder(itemView:View):Recyc
33          val imageView:ImageView = itemView.findViewByI
```

Oraz metodę onClick

ExampleAdapter.kt

```
27      //klasa odpowiedzialna za widoczność elementów listy
28      inner class ExampleViewHolder(itemView:View):RecyclerView.ViewHolder(
29          val imageView:ImageView = itemView.findViewById(R.id.image_view1)
30          val textView1:TextView = itemView.findViewById(R.id.text_view_1)
31          val textView2:TextView = itemView.findViewById(R.id.text_view_2)
32          init{
33              itemView.setOnClickListener(this)
34          }
35
36          override fun onClick(v: View?) {
37              val position = bindingAdapterPosition
38              if(position != RecyclerView.NO_POSITION) {
39                  listener.onItemClick(position)
40              }
41          }
42      }
43      interface OnItemClickListener {
44          fun onItemClick(position: Int)
45      }
46  }
```

W MainActivity.kt powinien pojawić się błąd przy zmiennej ‘adapter’ .

MainActivity.kt

```
14      private val adapter = ExampleAdapter(exampleList)
15      override fun onCreate(savedInstanceState: Bundle?) {
```

Dodajemy atrybut ‘this’:

```
13      private val exampleList = generuj( size: 500)
14      private val adapter = ExampleAdapter(exampleList,  listener: this)
15      override fun onCreate(savedInstanceState: Bundle?) {
```

Oraz dodajemy argument metody:

MainActivity.kt

```
2
3   import ...
10
11  class MainActivity : AppCompatActivity(), On {
12      lateinit var binding: ActivityMainB
13      private val exampleList = generuj(
```

OnErrorAction (kotlin.io)
OnItemClickListener (com.example.recy

Po dodaniu:

```
11  class MainActivity : AppCompatActivity(), ExampleAdapter.OnItemClickListener {
12      lateinit var binding: ActivityMainBinding
```

Implementujemy metodę w naszej aktywności

MainActivity.kt

```
1   package com.example.recyclerviewapp
2
3   import ...
10
11  class MainActivity : AppCompatActivity(), ExampleAdapter.OnItemClickListener {
12      lateinit var binding: ActivityMainBinding
13      private val exampleList = generuj( size: 500)
14      private val adapter = ExampleAdapter(exampleList,  listener: this)
15      override fun onCreate(savedInstanceState: Bundle?) {...}
25      fun insertItem(view:View){...}
34      fun removeItem(view:View){...}
39      // tutaj implememntujemy metodę onItemClick
40
41      private  fun generuj(size:Int):ArrayList<ExampleItem>{...}
50  }
```

```
39      // tutaj implememntujemy metodę onItemClick
40      ove
41      override
50  } override fun onItemClick(position: Int) {...}  OnItemClickListener
51      override fun applyOverrideConfiguration(overrideConfig…
```

MainActivity.kt

```
41    override fun onItemClick(position: Int) {
42        Toast.makeText( context: this,  text: "Element $position został kliknięty", Toast.LENGTH_SHORT).show()
43        val clickedItem =exampleList[position]
44    }
45    private  fun generuj(size:Int):ArrayList<ExampleItem>{...}
```

Teraz w klasie ExampleItem zmieniamy typ zmiennych na var

ExampleItem.kt

```
1  package com.example.recyclerviewapp
2
3  data class ExampleItem (val imageResource:Int,var text1:String, var text2:String)
```

Możemy jeszcze zmodyfikować wygląd kliknięty wybranych pozycji:

MainActivity.kt

```
40    override fun onItemClick(position: Int) {
41        Toast.makeText( context: this,  text: "Element $position został kliknięty", Toast.LENGTH_SHORT).show()
42        val clickedItem =exampleList[position]
43        clickedItem.text1 = "Element klimnięty"
44        clickedItem.text2 = "clicked"
45        adapter.notifyItemChanged(position)
46    }
```

Powinniśmy mieć możliwość zaobserwowania elementów kliknięty.

W celu usunięcia kliknięty pola można przerobić funkcję onItemClick:

```
40    override fun onItemClick(position: Int) {
41        Toast.makeText( context: this,  text: "Element $position został kliknięty", Toast.LENGTH_SHORT).show()
42        val clickedItem = exampleList[position]
43        //clickedItem.text1 = "Element klimnięty"
44        //clickedItem.text2 = "clicked"
45        exampleList.removeAt(position) //usuwa element
46        //adapter.notifyItemChanged(position)
47        adapter.notifyItemRemoved(position)
48
49    }
```

co zmieni zachowanie kliknięcia.