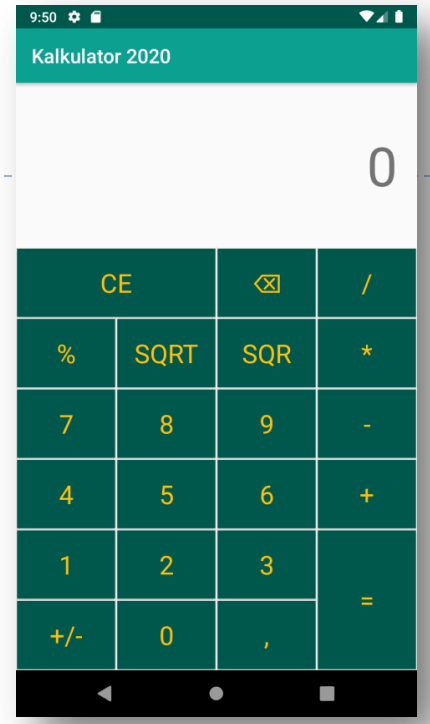
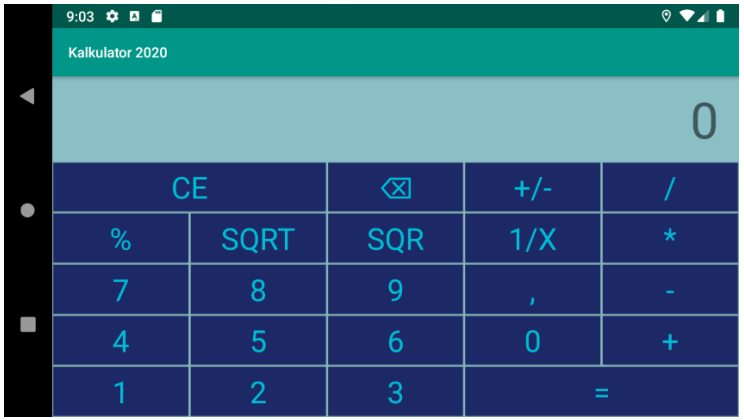
ZADANIE PRAKTYCZNE
KALKULATOR

Aplikacje mobilne



Stworzyć należy kalkulator wykonujący podstawowe działania:

- dodawanie,
- odejmowanie,
- dzielenie,
- mnożenie
- pierwiastek,
- kwadrat



Kalkulator powinien posiadać responsywny layout – dostosowujący się do orientacji ekranu

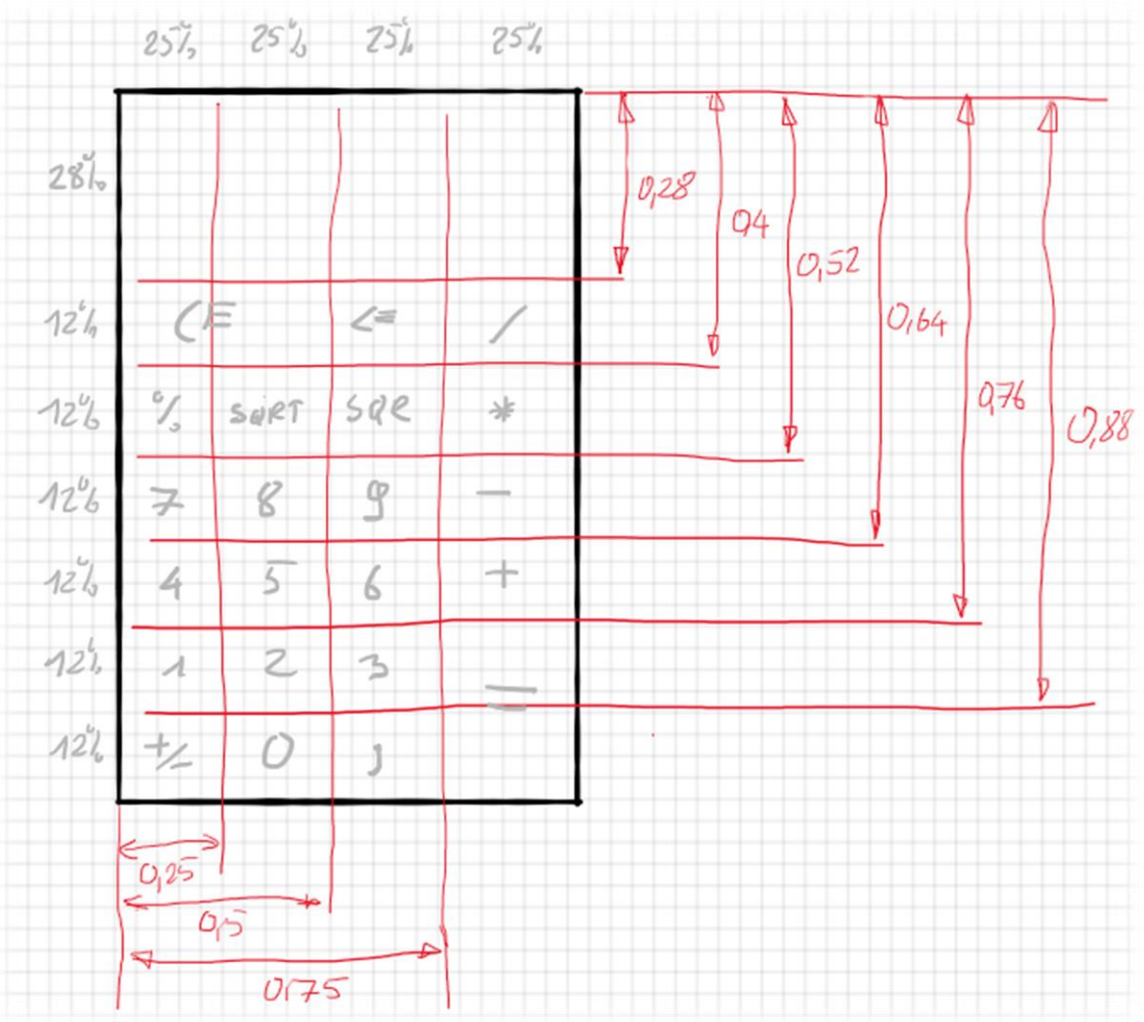
Tworzenie layout-u

Prace nad layoutem rozpoczynamy od przygotowania układu pionowego.

Układ layoutu oparty będzie na systemie linii pomocniczych

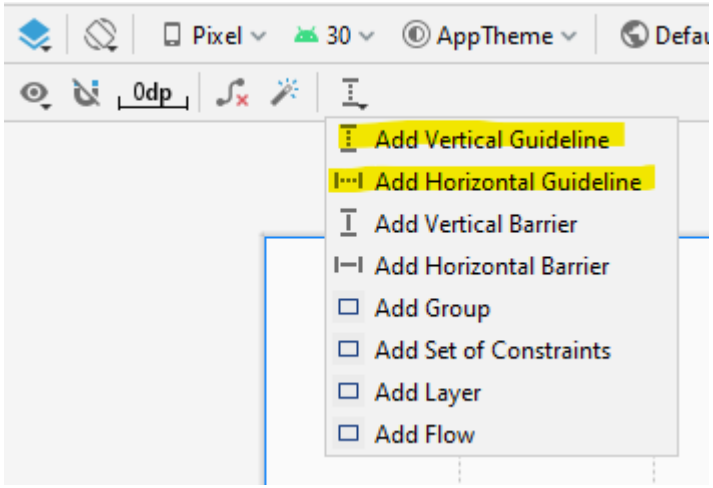
Ekran dzielimy na 8 wierszy z czego dwa górne przeznaczone będą na wyświetlacz. Potrzebujemy więc sześciu poziomych linii pomocniczych

Przyciski ułożone będą w czterech kolumnach. Potrzebować więc będziemy trzech pionowych linii pomocniczych.



Tworzenie layout-u

Linie pomocnicze najprościej dodać w zakładce designer.



Należy nadać własne ID dla linii.

W tym przypadku „gudelineH01”
(H oznacza horyzontal, 01 to numer kolejny)

Położenie tak utworzonej linii ustalone jest w jednostkach bezwzględnych dp.

```
app:layout_constraintGuide_begin="20dp"
```

W Naszym przykładzie lepiej zamieć je na wartość procentową

```
<androidx.constraintlayout.widget.Guideline
    android:id="@+id/guidelineH01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.25" />
```

Tworzenie layout-u

Łącznie potrzebujemy 10 linii pomocniczych.

Uzyskaliśmy siatkę w której rozmieszczać będziemy wszystkie elementy interfejsu.

Widok typu EditText będzie wyświetlaczem, a widoki typu Button – przyciskami.

```
9     <androidx.constraintlayout.widget.Guideline
10        android:id="@+id/guidelineH01"
11        android:layout_width="wrap_content"
12        android:layout_height="wrap_content"
13        android:orientation="horizontal"
14        app:layout_constraintGuide_percent="0.25" />
15
16    <androidx.constraintlayout.widget.Guideline...>
22
23    <androidx.constraintlayout.widget.Guideline...>
29
30    <androidx.constraintlayout.widget.Guideline...>
36
37    <androidx.constraintlayout.widget.Guideline...>
43
44    <androidx.constraintlayout.widget.Guideline...>
50
51    <androidx.constraintlayout.widget.Guideline...>
57
58    <androidx.constraintlayout.widget.Guideline...>
64
65    <androidx.constraintlayout.widget.Guideline...>
```

Tworzenie layout-u

Kolejnym krokiem jest przygotowanie schematu kolorystycznego naszej aplikacji. Schemat umieszczamy w pliku ***res/values/colors.xml***

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#009688</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="colorText">#FFC107</color>
    <color name="colorPrimary2">#8ABFC5</color>
    <color name="colorPrimaryDark2">#1D2967</color>
    <color name="colorText2">#00BCD4</color>
</resources>
```

Plik zawiera dwa schematy kolorystyczne – kolory o nazwach kończących się na „2” będą użyte do layoutu poziomego

Tworzenie layout-u

Widok typu TextView o ID=wyświetlacz będzie pokazywał wynik.

```
73      <TextView
74          android:id="@+id/wyswietlacz"
75          android:layout_width="0dp"
76          android:layout_height="0dp"
77          android:textAlignment="gravity"
78          android:gravity="right|center"
79          android:paddingRight="20dp"
80          android:text="0"
81          android:textSize="50sp"
82          app:layout_constraintBottom_toTopOf="@id/guidelineH01"
83          app:layout_constraintLeft_toLeftOf="parent"
84          app:layout_constraintRight_toRightOf="parent"
85          app:layout_constraintTop_toTopOf="parent" />
86
```

Tworzenie layout-u

Kolejnym krokiem jest przygotowanie przycisków.
Poniższy kod zawiera opis przycisku kasowania

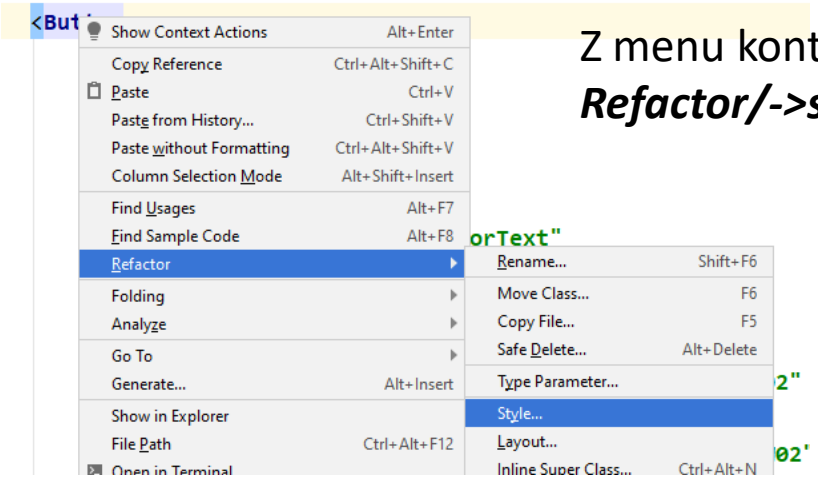
```
<Button
    android:id="@+id/kasuj"
    android:text="CE"
    android:layout_width ="0dp"
    android:layout_height="0dp"
    android:textSize="30sp"
    android:textColor="@color/colorText"
    android:background="@color/colorPrimaryDark"
    android:layout_margin="1dp"
    android:fontFamily="sans-serif"
    app:layout_constraintBottom_toTopOf="@id/guidelineH02"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="@id/guidelineW02"
    app:layout_constraintTop_toBottomOf="@id/guidelineH01" />
```

Warto zauważyć, że część parametrów (te zaznaczone żółtym markerem) będzie wspólna dla wszystkich przycisków.

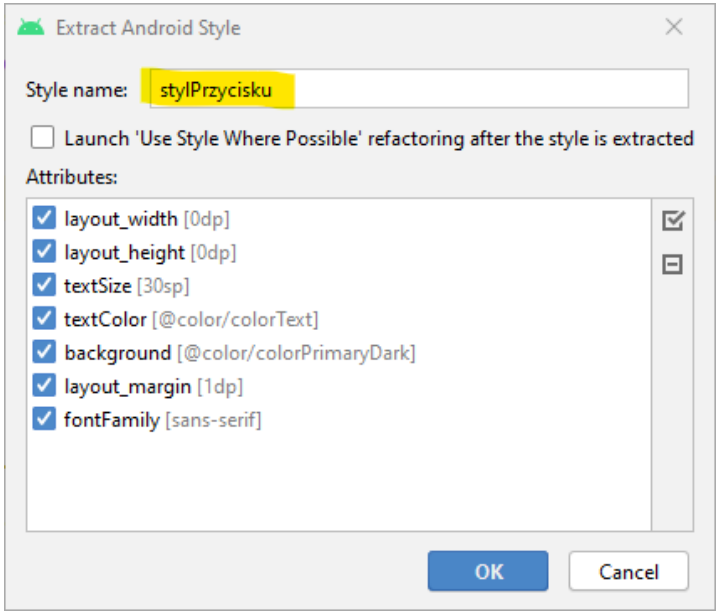
Powtarzające się parametry można wyodrębnić jako styl. A następnie dodać gotowy styl do wszystkich pozostałych przycisków. Rozwiązanie takie pozwala króćić kod, a także ułatwia późniejsze zmiany oraz modyfikacje.

Tworzenie stylu

Tworzenie stylu



Z menu kontekstowego wybieramy polecenie ***Refactor/->style…***



Nadajemy nazwę tworzonemu stylowi oraz wybieramy jakie parametry chcemy do niego wyeksportować.

Na liście nie pojawią się parametry takie jak ID, tekst czy pozycjonowanie. Z założenia są one indywidualne dla każdego widoku.

Tworzenie stylu

Style zapisywane są w pliku „styles.xml”.

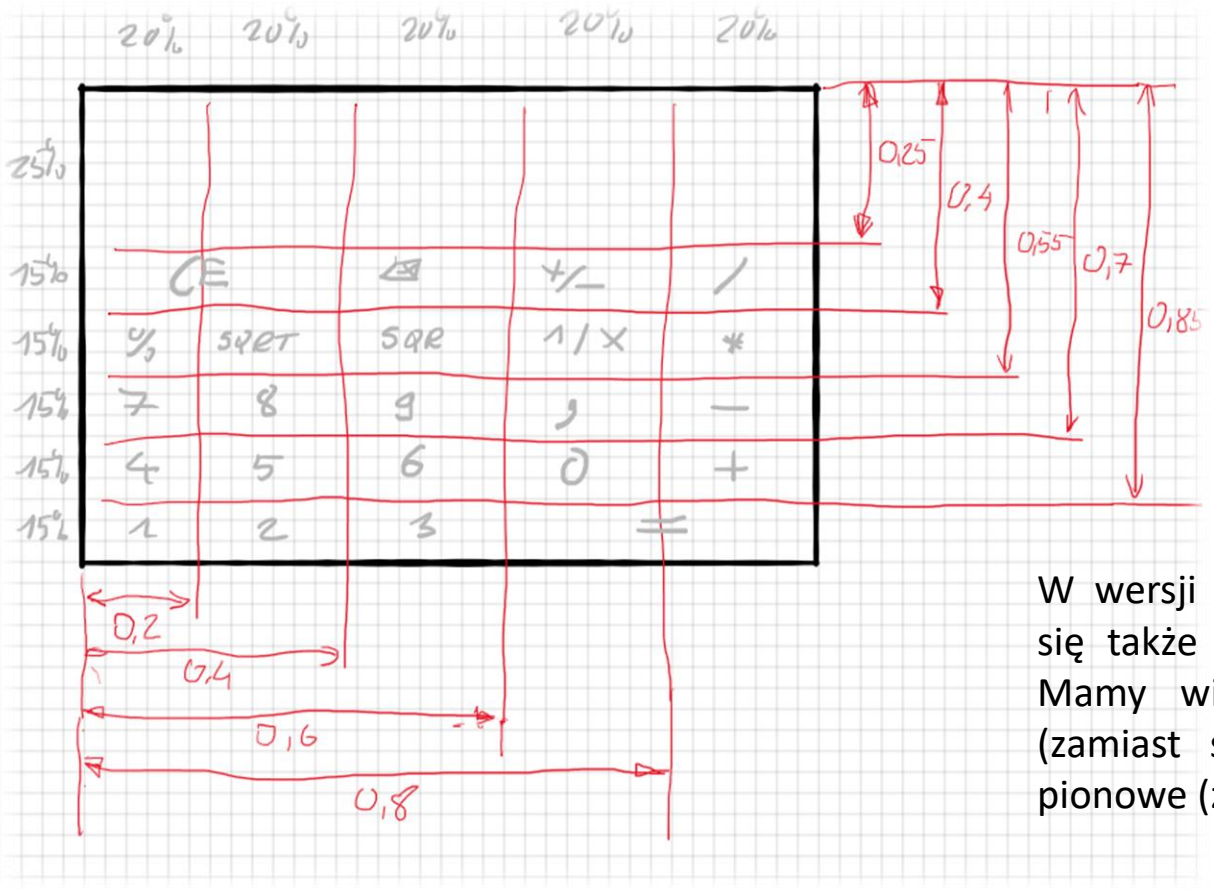
```
<style name="stylPrzycisku">
    <item name="android:layout_width">0dp</item>
    <item name="android:layout_height">0dp</item>
    <item name="android:textSize">30sp</item>
    <item name="android:textColor">@color/colorText</item>
    <item name="android:background">@color/colorPrimaryDark</item>
    <item name="android:layout_margin">1dp</item>
    <item name="android:fontFamily">sans-serif</item>
</style>
```

Po zastosowaniu stylu, kod przycisku wygląda w sposób następujący:

```
87  <Button
88      android:id="@+id/kasuj"
89      style="@style/stylPrzycisku"
90      android:text="CE"
91      app:layout_constraintBottom_toTopOf="@id/guidelineH02"
92      app:layout_constraintLeft_toLeftOf="parent"
93      app:layout_constraintRight_toRightOf="@id/guidelineW02"
94      app:layout_constraintTop_toBottomOf="@id/guidelineH01" />
```

Wersja pozioma layoutu

Tworzenie layoutu poziomego



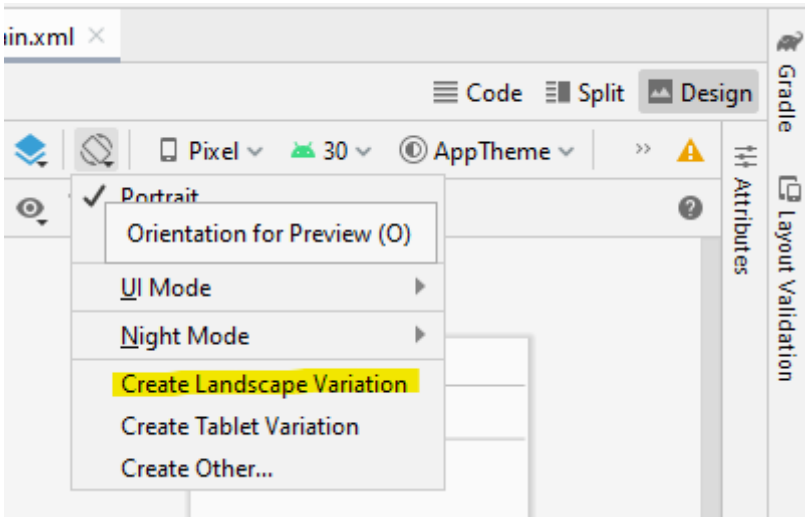
W poziomej wersji layoutu przyciski rozmieszczone będą w 5 rzędach po 5 kolumn.

Zauważmy że w tej wersji mamy 25 pól na przyciski zamiast 24 w wersji pionowej. Dzięki temu możliwe było rozbudowanie layoutu poziomego o dodatkową funkcję czyli odwrotność.

W wersji poziomej layoutu zmienia się także liczba linii pomocniczych. Mamy więc pięć linii poziomych (zamiast sześciu) oraz cztery linie pionowe (zamiast trzech).

Wersja pozioma layoutu

Tworzenie layoutu poziomego



Pamiętać należy że tworząc layout pionowy nie tworzymy od podstaw już istniejących przycisków, a tylko przesuwamy je w nowe miejsca.

Dotyczy to oczywiście tylko przycisków które powtarzają się na obu wersjach layoutu.

W tym wypadku dodać należy przycisk odwrotności, który layoucie pionowym nie występował.

Możliwe też jest usuwanie elementów, które występowały w wersji pionowej, a w poziomej nie są potrzebne.

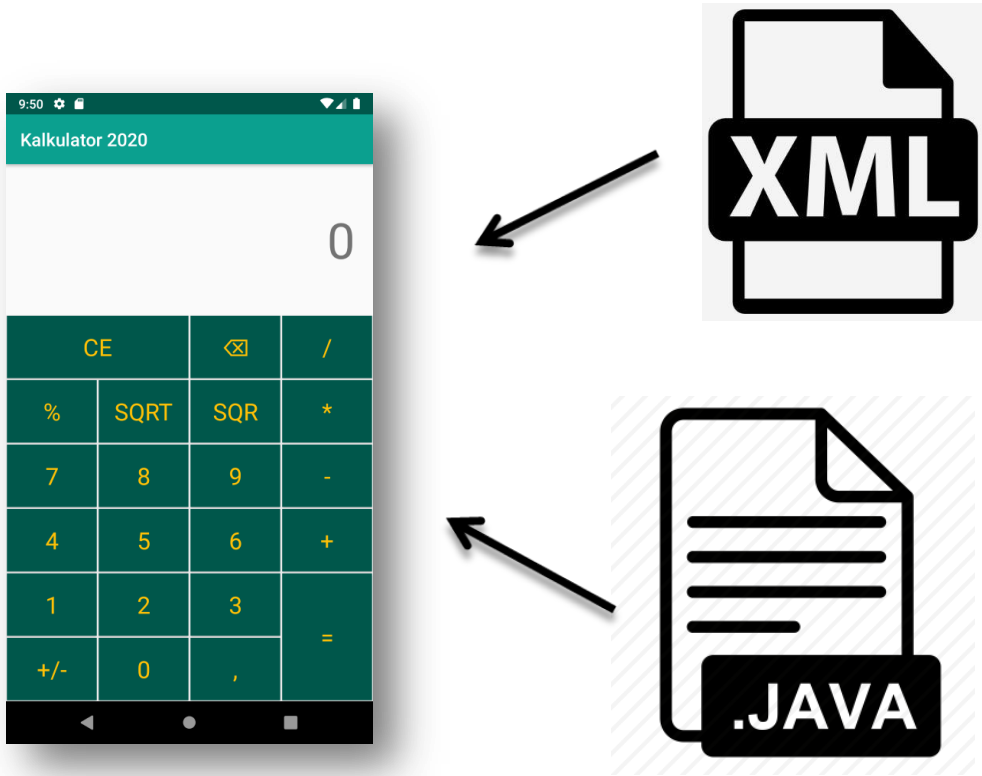
Aby dodać poziomy layout należy:

- otworzyć edytor z domyślnym layoutem,
- w prawym górnym roku kliknąć ikonę telefonu,
- z rozwiniętego menu wybrać opcję „*Create Landscape Variation*„

Utworzone zostaną dwa pliki ***activity_main.xml*** z tym, że jeden z nich oznaczony jest jako widok poziomy ***- land*** .

Aplikacje mobilne

Na tym etapie dysponujemy gotowym layoutem aplikacji i możemy przejść do pisania kodu Java, który zapewni jej działanie.



Kod Java

1. Powiązanie widoków z pliku .XML ze zmiennymi w programie

Rozpoczynamy od przygotowania zmiennych które będą przechowywać referencje do wszystkich kontrolek. Aby mieć do nich dostęp z poziomu wszystkich funkcji tworzymy je jako pola klasy reprezentującą główną (i jedyną) aktywność naszej aplikacji MainActivity.

```
19      Button p1, p2, p3, p4, p5, p6, p7, p8, p9, p0,
20              kasuj, wstecz, rowne, przecinek, zmianaZnaku,
21              procent, pierwiastek, kwadrat, odwrotnosc,
22              razy, podzielic, minus, plus;
23      TextView wyswietlacz;
```

Ze względu na dużą liczbę kontrolek odnalezienie referencji do nich oddelegujemy do osobnej funkcji ***odnajdzKontrolki( ),*** która uruchomiona zostanie na stracie aplikacji (w konstruktorze).

```
27      @Override
28  1   protected void onCreate(Bundle savedInstanceState) {
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.activity_main);
31          odnajdzKontrolki();
32      }
```

Kod Java -

Zauważmy że w naszym przykładzie nazwy zmiennych zawierających referencje do widoków są takie same jak ID tych widoków.

Nie jest to oczywiście żadna reguła lecz w programach posiadających dużą liczbę elementów może znacząco ułatwić tworzenie kodu.

```
36      private void odnajdzKontrolki() {
37          p1 = findViewById(R.id.p1);
38          p2 = findViewById(R.id.p2);
39          p3 = findViewById(R.id.p3);
40          p4 = findViewById(R.id.p4);
41          p5 = findViewById(R.id.p5);
42          p6 = findViewById(R.id.p6);
43          p7 = findViewById(R.id.p7);
44          p8 = findViewById(R.id.p8);
45          p9 = findViewById(R.id.p9);
46          p0 = findViewById(R.id.p0);
47          kasuj = findViewById(R.id.kasuj);
48          wstecz = findViewById(R.id.wstecz);
49          rowne = findViewById(R.id.rowne);
50          przecinek = findViewById(R.id.przecinek);
51          zmianaZnaku = findViewById(R.id.zmianaZnaku);
52          procent = findViewById(R.id.procent);
53          pierwiastek = findViewById(R.id.pierwiastek);
54          kwadrat = findViewById(R.id.kwadrat);
55          odwrotnosc= findViewById(R.id.odwrotnosc);
56          razy = findViewById(R.id.razy);
57          podzielic = findViewById(R.id.podzielic);
58          minus = findViewById(R.id.minus);
59          plus = findViewById(R.id.plus);
60          wyswietlacz = findViewById(R.id.wyswietlacz);
61          rowne = findViewById(R.id.rowne);
62          wstecz = findViewById(R.id.wstecz);
63      }
```

Kod Java – Słuchacz zdarzeń

Przyciski obsługiwane będą za pomocą słuchacza zdarzeń ***View.OnClickListener( )***. Możliwe są tu dwa podejścia: stworzenie kilku słuchaczy zależnie od rodzaju przycisku (inny słuchasz dla cyfr inny dla działań i tak dalej), lub też stworzę niejednego słuchacza zdarzeń obsługującego wszystkie przyciski. To drugie rozwiązanie jest czytelniejsze dlatego też to właśnie je zastosujemy w programie.

```
27      @Override
28      protected void onCreate(Bundle savedInstanceState) {
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.activity_main);
31          odnajdzKontrolki();
32          utworzSluchaczaZdarzen();
33      }
```

Słuchacz zdarzeń utworzony zostanie w metodzie ***utworzSluchaczaZdarzen( )*** która wywołana zostanie w konstruktorze klasy.

Kod Java – Słuchacz zdarzeń

Ze względu na sposób podziału naszego programu na metody konieczne jest utworzenie referencji do niego jako pola klasy (globalnie).

```
25      View.OnClickListener sluchaczZdarzen;
```

```
65      private void utworzSluchaczaZdarzen() {
66          sluchaczZdarzen = new View.OnClickListener() {
67              @Override
68              public void onClick(View view) {
69                  int id = view.getId();
70                  switch(id) {...}
95              }
96          };
97      }
```

Słuchacz odczytuje ID widoku na rzecz którego został wywołany i na tej podstawie wybiera odpowiednią akcję za pomocą funkcji switch( )

Uwaga: w metodzie tej tylko tworzymy słuchacza jeszcze nie przypisujemy go do żadnego widoku.

Kod Java
Słuchacz zdarzeń

```
27      @Override
28  1   protected void onCreate(Bundle savedInstanceState) {
29          super.onCreate(savedInstanceState);
30          setContentView(R.layout.activity_main);
31          odnajdzKontrolki();
32          utworzSluchaczaZdarzen();
33          dodajSluchaczeZdarzen();
34      }
```

Kolejnym krokiem jest dodanie słuchacza zdarzeń do wszystkich kontrolek (oprócz kontrolki TextView która nie musi reagować na kliknięcie).

```
99      private void dodajSluchaczeZdarzen() {
100         p1.setOnClickListener(sluchaczZdarzen);
101         p2.setOnClickListener(sluchaczZdarzen);
102         p3.setOnClickListener(sluchaczZdarzen);
103         p4.setOnClickListener(sluchaczZdarzen);
104         p5.setOnClickListener(sluchaczZdarzen);
105         p6.setOnClickListener(sluchaczZdarzen);
106         p7.setOnClickListener(sluchaczZdarzen);
107         p8.setOnClickListener(sluchaczZdarzen);
108         p9.setOnClickListener(sluchaczZdarzen);
109         p0.setOnClickListener(sluchaczZdarzen);
110         kasuj.setOnClickListener(sluchaczZdarzen);
111         plus.setOnClickListener(sluchaczZdarzen);
112         minus.setOnClickListener(sluchaczZdarzen);
113         razy.setOnClickListener(sluchaczZdarzen);
114         podzielic.setOnClickListener(sluchaczZdarzen);
115         rowne.setOnClickListener(sluchaczZdarzen);
116         wstecz.setOnClickListener(sluchaczZdarzen);
117         zmianaZnaku.setOnClickListener(sluchaczZdarzen);
118         przecinek.setOnClickListener(sluchaczZdarzen);
119         procent.setOnClickListener(sluchaczZdarzen);
120         kwadrat.setOnClickListener(sluchaczZdarzen);
121         pierwiastek.setOnClickListener(sluchaczZdarzen);
122     }
```

Kod Java - Słuchacz zdarzeń

Zauważmy, że lista kontrolek zmienia się zależnie od orientacji ekranu – na lyoucie poziomym pojawia się kontrolka „1/x” o ID=„odwrotnosc”

Próba odwołania się do kontrolki, która nie istnieje na aktualnym layoucie spowoduje błąd.

W tym przypadku wystąpi on już na etapie dodawania słuchacza zdarzeń.

Rozwiązaniem jest sprawdzenie aktualnej orientacji ekranu i dodanie słuchacza tylko wtedy, gdy tworzony jest layout poziomy (zawierający tą kontrolkę)

```
121     if(getResources().getConfiguration().orientation == Configuration.ORIENTATION_LANDSCAPE)
122         odwrotnosc.setOnClickListener(sluchaczZdarzen);
```

Kod Java – dodawanie cyfr

Pierwszą funkcjonalnością którą należy o programować jest obsługa przycisków dodających cyfry.

Każde kliknięcie w przycisk spowoduje dopisanie odpowiedniej cyfry do łańcucha znaków przechowywanego w kontrolce ***Tekst View*** o referencji ***wyswietlacz.***

Wykona to metoda ***dodajCyfre( ),*** która otrzyma w parametrze znak który należy dopisać.

Zauważmy jednak, że kliknięcie w kalkulatorze na przycisk z cyfrą nie zawsze daje ten sam efekt.

- Jeżeli jesteśmy w trakcie wpisywania liczby - cyfra zostanie po prostu dopisana na ekran.
- Jeżeli wcześniej kliknięty był znak równości lub znak działania rozpoczynamy wpisywanie nowej liczby, a w takim przypadku ekran powinien być najpierw skasowany.

Do rozpoznawania, w którym trybie wpisywanie jesteśmy (czyli jak ma zachować się przycisk) posłuży nam ***flaga*** czyli globalna zmienna boolowska o nazwie ***trybPisania***.

```
16        boolean trybPisania = true;
```

- Wartość ***true*** oznacza, że jesteśmy w trakcie wpisywania liczby i każdorazowe kliknięcie przycisku dodaje nową cyfrę.
- Wartość ***false*** oznaczała będzie, że rozpoczynamy wpisywanie nowej liczby, czyli zawartość kontrolki textView ma być skasowana i dopiero wtedy dodajemy wybraną cyfrę.

Kod Java – dodawanie cyfr

```
16      boolean trybPisania = true;
```

```
124     private void dodajCyfre(String s) {
125         if (trybPisania) {
126             String temp = wyswietlacz.getText().toString();
127             if (temp.equals("0")) temp="";
128             wyswietlacz.setText(temp+s);
129         }
130         else
131         {
132             wyswietlacz.setText(s);
133             trybPisania=true;
134         }
135     }
```

W trybie pisania: pobieramy aktualną zawartość wyświetlacza do zmiennej temp, a następnie dopisujemy do niej znak przekazany w parametrze i wysyłamy spowrotem do kontrolki TextView (wyświetlacz).

Warunek ***if(temp.equals(„0”))*** zapobiega wpisaniu kliku zer na początku liczby (jak np. 0005).

W trybie rozpoczynania nowej liczby: zastępujemy łańcuch znaków przechowywany w kontrolce TextView cyfrą przekazaną w parametrze oraz ustawiamy flagę ***trybPisania=tru***e; (aby możliwe było dodawanie kolejnych cyfr).

Kod Java – dodawanie cyfr

Wywołanie metody ***dodajCyfre( )*** następuje w słuchaczu zdarzeń. Parametr przekazywany do metody zależy, od tego który przycisk ją wywołał.

```
65      private void utworzSluchaczaZdarzen() {
66          sluchaczZdarzen = new View.OnClickListener() {
67              @Override
68              public void onClick(View view) {
69                  int id = view.getId();
70                  switch(id) {
71                      case R.id.p1 : dodajCyfre( s: "1"); break;
72                      case R.id.p2 : dodajCyfre( s: "2"); break;
73                      case R.id.p3 : dodajCyfre( s: "3"); break;
74                      case R.id.p4 : dodajCyfre( s: "4"); break;
75                      case R.id.p5 : dodajCyfre( s: "5"); break;
76                      case R.id.p6 : dodajCyfre( s: "6"); break;
77                      case R.id.p7 : dodajCyfre( s: "7"); break;
78                      case R.id.p8 : dodajCyfre( s: "8"); break;
79                      case R.id.p9 : dodajCyfre( s: "9"); break;
80                      case R.id.p0 : dodajCyfre( s: "0"); break;
81                  }
82              }
83          };
84      }
```

Kod Java – usuwanie ostatniej cyfry (cofanie)

```
186     private void cofnij() {
187         String s = wyswietlacz.getText().toString();
188         if (s.length()>1) {
189             s = s.substring(0,s.length()-1);
190         }
191         else s="0";
192         wyswietlacz.setText(s);
193     }
```

1. Pobieramy aktualną zawartość wyświetlacza do zmiennej **s**.
2. Jeżeli długość łańcucha **s** jest większa niż jeden (na ekranie są jeszcze cyfry do skasowania) tworzymy substring (podłańcuch) o jeden znak krótszy od łańcucha **s** i zastępujemy nim łańcuch **s**. W efekcie obcinamy ostatni znak.
3. Jeżeli łańcuch **s** już pusty (ma zerową długość) zastępujemy go łańcuchem **„0”** – skasowaliśmy całą liczbę, więc na ekranie powinno się wyświetlać **„0”**
4. Wypisujemy łacnych **s** na ekran**.**

Wywołanie metody należy dodać do słuchacz zdarzeń

```
case R.id.kasuj : kasuj(); break;
```

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)

```
65      private void utworzSluchaczaZdarzen() {
66          sluchaczZdarzen = new View.OnClickListener() {
67              @Override
68              public void onClick(View view) {
69                  int id = view.getId();
70                  switch(id) {
71                  // ..................
72                      case R.id.plus : dzialanie( kodPrzycisku: 1); break;
73                      case R.id.minus : dzialanie( kodPrzycisku: 2); break;
74                      case R.id.razy : dzialanie( kodPrzycisku: 3); break;
75                      case R.id.podzielic : dzialanie( kodPrzycisku: 4); break;
76                      case R.id.rowne : dzialanie( kodPrzycisku: 0); break;
77                  //..................
78                  }
79              }
80          };
81      }
```

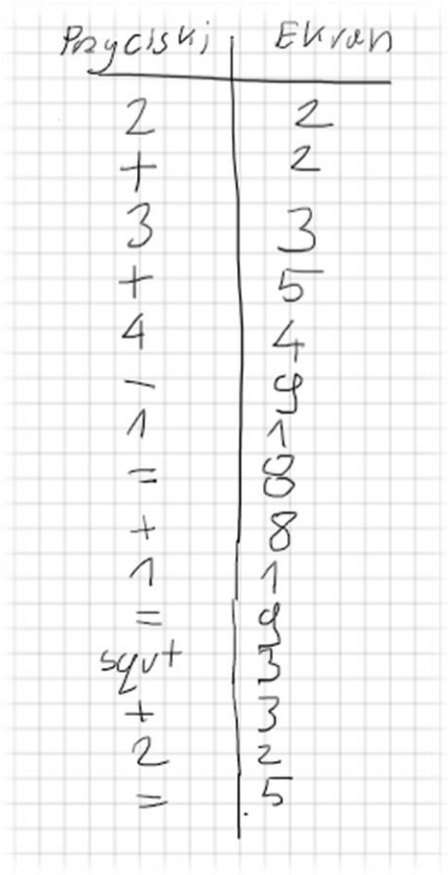
Za obsługę operatorów działań wieloargumentowych (+ ; - ; *; /) oraz „=„ odpowiadać będzie metoda ***dzialanie( ).***

Wyboru działania dokonujemy na podstawie przypisanych im numerów.

1 – dodawanie,
2 – odejmowanie,
3 – mnożenie,
4 – dzielenie.
0 – znak „=„

Znak „=„ to nie operator działania, więc traktowany jest inaczej, ale także obsługiwany przez metodę ***dzialanie( )***

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)



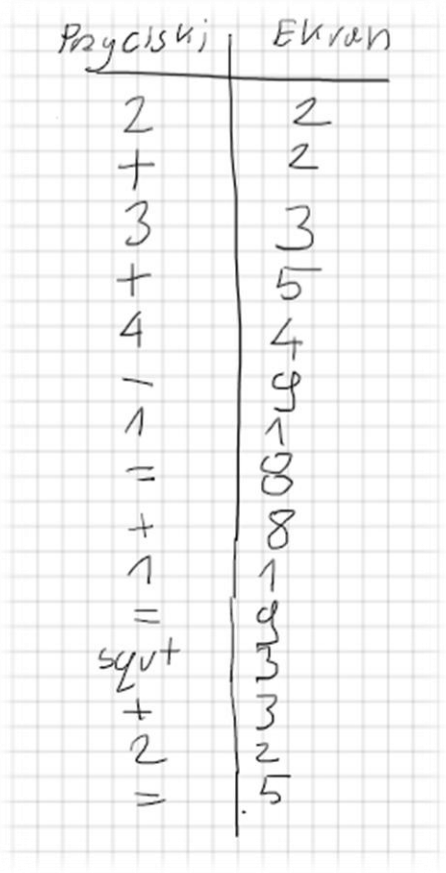
Nasz kalkulator potrafi wykonywać tylko podstawowe obliczenia arytmetyczne, jednak nawet w tym przypadku jego działanie nie jest aż takie proste jak mogłoby się to wydawać.

Praca z kalkulatorem rzadko ogranicza się do dwuargumentowy działań w rodzaju 2+2=4. Zwykle mamy do czynienia z ciągiem działań na (przykład wielokrotne dodawanie).

Aby zrozumieć problem przeanalizujmy przykład pracy na kalkulatorze (na kalkulatorze systemowym w Windows).

Po lewej stronie tabelki widzimy jakie przyciski naciskano, a po prawej widzimy to, co było w tym momencie wyświetlane na ekranie.

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)



Z analizy tego przykładu można wyciągnąć kilka wniosków:

- ✓ Jeżeli znak działania występuje w ciągu działań, powoduje dwie operacje.
  - Po pierwsze - kończy poprzednie działanie i wyświetla wynik.
  - Po drugie – zapamiętuje, jakie będzie następne działanie do wykonania i czeka na drugi argument tego działania.
- ✓ Inaczej zachowuje się program gdy przycisk działania jest użyty po raz pierwszy w ciągu działań (rozpoczyna nowy ciąg). Wtedy kliknięcie przycisku nie zmienia liczby wyświetlanej na ekranie, a tylko powoduje zapamiętanie wybranego działania.
- ✓ Znak równości powoduje dokończenie rozpoczętego obliczenia, wyświetla wynik oraz kończy ciąg działań. Następne użycie przycisku operatora działania będzie oznaczało rozpoczęcie nowego ciągu, gdzie pierwszym argumentem jest liczba aktualnie wyświetlana na ekranie
- ✓ Operatory działań jednoargumentowy, takich jak pierwiastek, kwadrat, procent oraz zmiana znaku, modyfikują tylko wartość wyświetlaną aktualnie na ekranie. Reakcja na ich naciśnięcie jest natychmiastowa (nie wymagają znaku równości ani nie oczekują na kolejny argument).
- ✓ Operatory jednoargumentowe nie kończą ciągu operacji.

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)

```
13      double x1=0, x2=0;
14      int kodDzialania =0;
15
16      boolean trybPisania = true,
17              pierwszeDzilanie = true;
```

Zauważmy że po naciśnięciu przycisku działania kończone jest działanie poprzednie.

Metoda obsługi przycisku potrzebuje więc dwóch informacji:

- kodu aktualnego działania (aktualnie naciśnięty przycisk), który otrzymuje w parametrze (***kodPrzycisku***)
- kodu poprzedniego działania (***kodDzilania***), który musi być polem globalnym klasy, gdyż jest przechowywany od poprzedniego użycia metody działanie.

Istotną jest też informacja o tym, czy jest to pierwsze działanie w ciągu, czy też jedno z kolejnych. Informację taką będziemy przechowywać w polu typu boolean o nazwie ***pierwszeDzialanie.***

Zmienne typu rzeczywistego ***x1, x2*** będą przechowywały argumenty wykonywanego działania. ***x1*** musi być polem klasy gdyż jej zadaniem jest zapamiętanie wartości pierwszego argumentu wykonywanego działania w czasie, gdy na ekran wpisywany jest drugi argument.

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)

```
137     private void dzialanie(int kodPrzycisku) {
138         double wynik=0;
139         if (pierwszeDzilanie)
140         {
141             x1= Double.parseDouble(wyswietlacz.getText().toString());
142             pierwszeDzilanie=false;
143         }
144         else
145         {
146             x2 = Double.parseDouble(wyswietlacz.getText().toString());
147             switch (kodDzialania)
148             {
149                 case 1: wynik = x1+x2; break;
150                 case 2: wynik = x1-x2; break;
151                 case 3: wynik = x1*x2; break;
152                 case 4: wynik = x1/x2; break;
153             }
154             x1=wynik;
155             wyswietlacz.setText(String.valueOf(wynik));
156         }
157         kodDzialania= kodPrzycisku;
158         if (kodPrzycisku==0) {
159             pierwszeDzilanie=true;
160         }
161         trybPisania=false;
162     }
```

Objaśnienia na następnej stronie

Kod Java – obsługa działań wieloargumentowych (+,-,*,/,=)

1. **Jeżeli jest to pierwsze działanie w ciągu (pierwszeDzialanie==true):** w zmiennej ***x1*** zapamiętujemy liczbę z wyświetlacza (pierwszy argument działania), następnie ustawiamy flagę ***pierwszeDzialanie = false***, gdyż kolejne kliknięcie przycisku działania będzie już drugą operacją w ciągu.
2. ***Jeżeli wykonywane działanie nie rozpoczyna nowego ciągu operacji*** **(pierwszeDzialanie==false):** Stan wyświetlacza zapisujemy w zmiennej ***x2*** jako drugi argument działania. Następnie, na podstawie pola ***kodDzialania*** przechowującego kod poprzednio wybranego działania (tego które właśnie musimy zakończyć) wykonujemy odpowiednią operacja arytmetyczną na obu argumentach działania (***x1, x2***). Wynik zapamiętujemy w zmiennej lokalnej ***wynik***. Następnie ***wynik*** przepisujemy do pola ***x1,*** gdyż może on stać się pierwszym argumentem w następnym działaniu. Na koniec wypisujemy ***wynik*** na wyświetlacz.
3. ***Niezależnie od tego czy jest to pierwsze czy kolejne działanie w ciągu:*** zapamiętujemy, do przyszłego wykorzystania, kod przycisku przepisując go do pola ***kodDzialania*** . Flagę ***trybPisania*** ustawiamy na ***false*** aby umożliwić rozpoczęcie wpisywania nowej liczby.
4. Jeżeli wybrano znak równości (***kodPrzycisku==0***) nie wykonujemy żadnych obliczeń, lecz przerywamy ciąg operacji ustawiamy wartość zmiennej ***pierwszeDzialanie*** na ***true,*** co sprawi, że następna operacja potraktowana będzie jako otwierająca nowy ciąg.

Kod Java – formatowanie wyniku przed wypisaniem na ekran

Proste wypisywanie na ekran wyniku przechowywanego w zmiennej typu double powoduje pewne problemy.

Najbardziej widoczne z nich to:

- zbyt duża liczba miejsc po przecinku powodująca, że liczba nie mieści się na wyświetlaczu
- dopisywanie końcówki „ . 0” do wyników będących liczbami całkowitymi

Rozwiązaniem jest stworzenie metody która przetworzy wynik przed jego wyświetleniem na ekranie.

```
203     private void wyswietl(double w) {
204         w = Math.round(w*1000000)/1000000.0;
205         String s = String.valueOf(w);
206         if (s.substring(s.length()-2).equals(".0")) s= s.substring(0, s.length()-2);
207         wyswietlacz.setText(s);
208     }
```

Powyższa metoda wykonuje dwie operacje:

zaokrągla wynik do 8 miejsc po przecinku.

Pozbywa się końcówki „ .0” - przekazana w parametrze liczba jest zamieniona na łańcuch. Jeżeli kończy się on na „ .0” obcinane są dwa ostatnie nadmiarowe znaki. Dopiero tak przetworzony łańcuch wypisywany jest na wyświetlacz.

```
wyswietlacz.setText(String.valueOf(wynik));   ⇒   wyswietl(wynik);
```

Kod Java – działania jednoargumentowe (sqr sqrt +/- 1/x)

Podobnie jak działania dwuargumentowe, działania jednoargumentowe obsługiwane będą przez wspólną metodę.

Będzie ona wywoływana w słuchaczu zdarzeń z odpowiednim parametrem (zależnie, który przycisk nacisnęliśmy)

1- procent,

2- pierwiastek,

3 - kwadrat,

4 – odwrotność,

```
case R.id.procent : funkcja( kodPrzycisku: 1); break;
case R.id.pierwiastek : funkcja( kodPrzycisku: 2); break;
case R.id.kwadrat : funkcja( kodPrzycisku: 3); break;
case R.id.odwrotnosc : funkcja( kodPrzycisku: 4); break;
```

Kod Java – działania jednoargumentowe (sqr sqrt +/- 1/x)

```
150     private void funkcja(int kodPrzycisku) {
151         double  wynik=0;
152         Double x = Double.parseDouble(wyswietlacz.getText().toString());
153         switch (kodPrzycisku) {
154             case 1: wynik = x*100; break;
155             case 2: wynik = Math.sqrt(x); break;
156             case 3: wynik = x*x; break;
157             case 4: wynik= 1/x; break;
158         }
159         wyswietl(wynik);
160         trybPisania=false;
161     }
```

Obsługa działań jednoargumentowy ich jest prostsza niż w przypadku działań dwuargumentowy.

Metoda ***funkcja( )*** otrzymuje w parametrze ***kodPrzycisku.*** Do zmiennej x pobiera wartość z wyświetlacza, a następnie wykonuje właściwe obliczenie na podstawie kodu przycisku i wynik wypisuje na wyświetlacz.

Pamiętać należy że po wypisaniu wyniku ***trybPisania*** przełączyć należy na ***false*** aby umożliwić rozpoczęcie wpisywania nowej liczby.

Kod Java – zmiana znaku (+/-)

```
179     private void zmienZnak() {
180         String s=wyswietlacz.getText().toString();
181         double w = Double.valueOf(s);
182         wyswietl(-w);
183     }
```

Zmiana znaku na przeciwny polega na:

- pobraniu zawartości wyświetlacza,
- przekonwertowaniu jej na wartość double – zapisaną w zmiennej ***w,***
- wypisaniu na ekran wartości -***w***

Kod Java – kasowanie ekranu

```
163     private void kasuj() {
164         wyswietlacz.setText("0");
165         pierwszeDzilanie=true;
166         kodDzialania=0;
167         trybPisania=true;
168     }
```

Przycisk kasowania powoduje zastąpienie wartości aktualnej przechowywanej na wyświetlaczu łańcuchem „0”.

Pamiętać należy, że skasowanie ekranu jest przerwaniem aktualnie wykonywanego ciągu operacji i przygotowaniem do rozpoczęcia następnego. Konieczne jest więc:

przestawienie flagi ***trybPisania*** na ***true*** - będziemy wpisywać nową liczbę,

ustawienie flagi ***pierwszeDzialanie*** na ***true*** - rozpoczynamy nowy ciąg operacji.

Kod Java – kasowanie ostatniego znaku (cofanie)

```
170     private void cofnij() {
171         String s = wyswietlacz.getText().toString();
172         if (s.length()>1) {
173             s = s.substring(0,s.length()-1);
174         }
175         else s="0";
176         wyswietlacz.setText(s);
177     }
```

Usuwanie ostatniego znaku (cofanie) polega na:

- pobraniu z ekranu przechowywanego na nim łańcucha znaków
- stworzenie substring-u krótszego o jeden znak i wypisanie tego skróconego łańcucha na ekran (obcięcie ostatniego znaku)

Przed wykonaniem tej operacji należy jednak upewnić się, że łańcuch zawiera więcej niż jeden znak.

Jeżeli łańcuch zawiera tylko jeden znak, to po skróceniu go stanie się łańcuchem pustym. Jednak na ekran zamiast niego wypisać należy wartość „***0***”

Kod Java – dodawanie przecinka

```
185     private void wstawPrzecinek() {
186         if (trybPisania)
187         {
188             String s = wyswietlacz.getText().toString();
189             if (s.indexOf('.')<0) {
190                 s+=".";
191                 wyswietlacz.setText(s);
192             }
193         }
194         else
195         {
196             wyswietlacz.setText("0.");
197             trybPisania=true;
198         }
```

Dodając przecinek należy sprawdzić czy, nie występuje on już we wpisywanej liczbie. Liczba z dwoma przecinkami byłaby błędna.

W tym celu posłużyć się można metodą ***.indexOF( )*** wywołaną dla pobranego z wyświetlacza łańcucha znaków. Metoda ta zwróci wartość ujemną, jeżeli nie znajdzie podanego w parametrze znaku. Wtedy do zmiennej ***s*** można dopisać przecinek i wyświetlić ją na ekran.

Przecinek dopisujemy gdy jesteśmy w trybie pisania (***trybPisania==true***) czyli kontynuujemy wpisywanie liczby. Jeżeli dopiero rozpoczynamy nową liczbę nie może się ona zaczynać od przecinka. W takim przypadku należy wypisać *„**0.**”*

Przecinek na końcu liczby jest dopuszczalny. Nie wpływa on na wynik obliczeń i nie generuje błędu.

Słuchacz zdarzeń – efekt końcowy

Wszystkie utworzone metody należy dodać do słuchacza zdarzeń (podpiąć pod odpowiednie widoki)

```
65      private void utworzSluchaczaZdarzen() {
66          sluchaczZdarzen = new View.OnClickListener() {
67              @Override
68              public void onClick(View view) {
69                  int id = view.getId();
70                  switch(id) {
71                      case R.id.p1 : dodajCyfre( s: "1"); break;
72                      case R.id.p2 : dodajCyfre( s: "2"); break;
73                      case R.id.p3 : dodajCyfre( s: "3"); break;
74                      case R.id.p4 : dodajCyfre( s: "4"); break;
75                      case R.id.p5 : dodajCyfre( s: "5"); break;
76                      case R.id.p6 : dodajCyfre( s: "6"); break;
77                      case R.id.p7 : dodajCyfre( s: "7"); break;
78                      case R.id.p8 : dodajCyfre( s: "8"); break;
79                      case R.id.p9 : dodajCyfre( s: "9"); break;
80                      case R.id.p0 : dodajCyfre( s: "0"); break;
81                      case R.id.kasuj : kasuj(); break;
82                      case R.id.plus : dzialanie( kodPrzycisku: 1); break;
83                      case R.id.minus : dzialanie( kodPrzycisku: 2); break;
84                      case R.id.razy : dzialanie( kodPrzycisku: 3); break;
85                      case R.id.podzielic : dzialanie( kodPrzycisku: 4); break;
86                      case R.id.rowne : dzialanie( kodPrzycisku: 0); break;
87                      case R.id.wstecz : cofnij(); break;
88                      case R.id.zmianaZnaku : zmienZnak(); break;
89                      case R.id.przecinek : wstawPrzecinek(); break;
90                      case R.id.procent : funkcja( kodPrzycisku: 1); break;
91                      case R.id.pierwiastek : funkcja( kodPrzycisku: 2); break;
92                      case R.id.kwadrat : funkcja( kodPrzycisku: 3); break;
93                      case R.id.odwrotnosc : funkcja( kodPrzycisku: 4); break;
94                  }
95              }
96          };
```