

Mobilne Aplikacije

Nositelj: doc. dr. sc. Nikola Tanković

Izvođač: dr. sc. Robert Šajina

Asistent: mag. inf. Alesandro Žužić

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

[5] – Dinamičko definiranje prikaza

Posljednje ažurirano: 10. prosinca 2025.

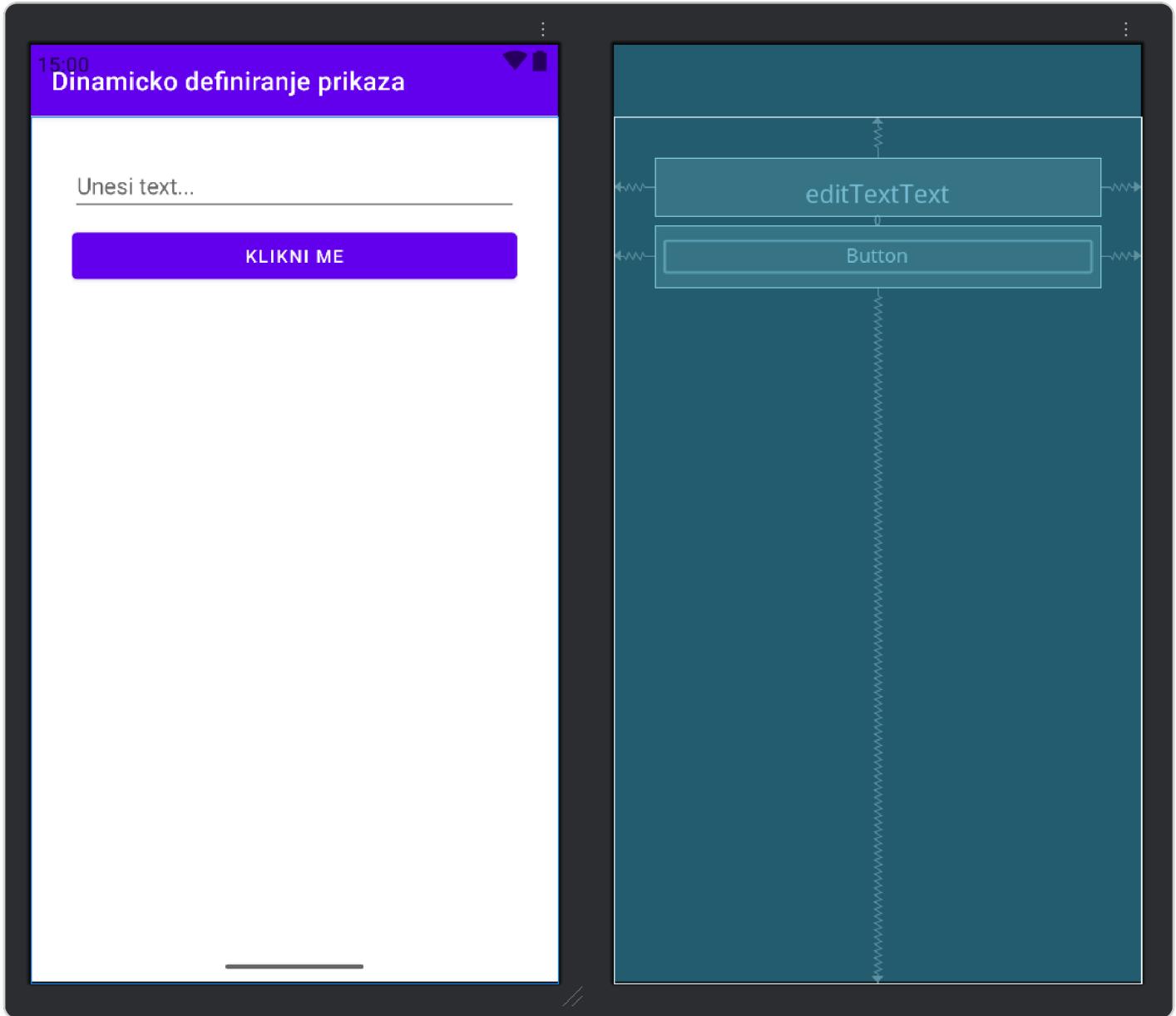
Sadržaj

- [Sadržaj](#)
- [Programsko definiranje prikaza](#)
- [Programska izmjena prikaza](#)
- [Definiranje vlastitog prikaza](#)
 - [Koristeći postojeće prikaze](#)
 - [Samostalna izrada prikaza](#)

Programsko definiranje prikaza

U prethodnom poglavlju smo vidjeli kako definirati prikaz korištenjem XML datoteka. Međutim, moguće je definirati prikaz i programskim putem, tj. u samom kôdu aktivnosti ili fragmenta. Ovakav pristup može biti koristan kada želimo dinamički mijenjati izgled korisničkog sučelja na temelju određenih uvjeta ili podataka.

Primjerice, možemo stvoriti jednostavnu aktivnost koja programskim putem definira nekoliko elemenata i postavlja ih unutar `ConstraintLayout`. Izgledala bi kao na slici:



GUI za rekreirati

Gdje mi želimo programskim putem definirati `EditText` i `Button` unutar `ConstraintLayout` da bude posloženo kao u xml datoteci.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="32dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editTextText"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:ems="10"
        android:hint="Unesi text..."
        android:inputType="text"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        app:layout_constraintVertical_chainStyle="packed" />

    <Button
        android:id="@+id/button"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Klikni me"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editTextText" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Prvo ćemo u `onCreate` metodi naše aktivnosti definirati `ConstraintLayout` i postaviti ga kao glavni prikaz aktivnosti. Zatim ćemo stvoriti `EditText` i `Button`, postaviti njihove atribute i dodati ih u `ConstraintLayout`. Na kraju, definirat ćemo ograničenja za pozicioniranje elemenata unutar `ConstraintLayout`.

1. Kreiramo `ConstraintLayout` i postavljamo ga kao glavni prikaz aktivnosti:

```

...
ConstraintLayout mainLayout = new ConstraintLayout(this);
setContentView(mainLayout);
...

```

```
setContentView() uvijek mora biti pozvan nakon što smo definirali glavni layout.
```

2. Postavimo atribute za `ConstraintLayout` koristeći `setLayoutParams()` metodu:

```
...
mainLayout.setLayoutParams(new ViewGroup.LayoutParams(
    ViewGroup.LayoutParams.MATCH_PARENT,
    ViewGroup.LayoutParams.MATCH_PARENT)
);
...
```

Za postavljanje parametara izgleda koristimo `ViewGroup.LayoutParams` kako bismo definirali širinu i visinu `ConstraintLayout-a`. Da bismo dodali padding, koristimo `setPadding()` metodu:

```
...
int mainPaddingDp = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 32,
getResources().getDisplayMetrics());
mainLayout.setPadding(mainPaddingDp, mainPaddingDp, mainPaddingDp, mainPaddingDp);
...
```

Ovdje koristimo `TypedValue.applyDimension()` kako bismo pretvorili vrijednost paddinga iz dp u piksela, što je potrebno za pravilno postavljanje paddinga na različitim uređajima s različitim gustoćama ekrana.

3. Kreiramo `EditText` i postavljamo njegove atribute:

```
...
EditText editText = new EditText(this);

editText.setHint("Unesi text...");
editText.setId(EditText.generateViewId());
editText.setInputType(InputType.TYPE_CLASS_TEXT);

editText.setLayoutParams(new ViewGroup.LayoutParams(
    0, ViewGroup.LayoutParams.WRAP_CONTENT)
);

mainLayout.addView(editText);
...
```

Koristimo `EditText.generateViewId()` kako bismo generirali jedinstveni ID za `EditText`, što je važno za postavljanje ograničenja u `ConstraintLayout`.

4. Kreiramo Button i postavljamo njegove atribute:

```
...
Button button = new Button(this);

button.setText("Klikni me");
button.setId(Button.generateViewId());

button.setLayoutParams(new ViewGroup.LayoutParams(
    0, ViewGroup.LayoutParams.WRAP_CONTENT)
);
mainLayout.addView(button);
...
```

5. Definiramo ograničenja za pozicioniranje elemenata unutar ConstraintLayout:

```
...
ConstraintSet set = new ConstraintSet(); // Stvaramo constraint set
set.clone(mainLayout); // Kopiramo trenutne atribute iz constraint layout-a

// Constraint za EditText
set.connect(editText.getId(), ConstraintSet.START, ConstraintSet.PARENT_ID,
ConstraintSet.START);
set.connect(editText.getId(), ConstraintSet.END, ConstraintSet.PARENT_ID,
ConstraintSet.END);
set.setVerticalBias(editText.getId(), 0);

// Constraint za Button
set.connect(button.getId(), ConstraintSet.START, ConstraintSet.PARENT_ID,
ConstraintSet.START);
set.connect(button.getId(), ConstraintSet.END, ConstraintSet.PARENT_ID,
ConstraintSet.END);
```

Koristimo ConstraintSet kako bismo definirali veze između elemenata i roditeljskog ConstraintLayout.

6. Stvaramo vertikalni lanac kako bismo postigli željeni raspored elemenata:

```
// Chain
set.createVerticalChain(
    ConstraintSet.PARENT_ID, ConstraintSet.TOP,
    ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM,
    new int[]{editText.getId(), button.getId()}, //definira redoslijed elemenata u
lanцу
    null,
    ConstraintSet.CHAIN_PACKED);

set.applyTo(mainLayout);
...
```

`createVerticalChain()` metoda omogućuje nam da definiramo lanac elemenata unutar `ConstraintLayout`, što nam pomaže u postizanju željenog rasporeda.

7. Na kraju, dodajemo kôd za pravilno rukovanje sistemskim prozorima:

```
setContentView(mainLayout);
ViewCompat.setOnApplyWindowInsetsListener(mainLayout, (v, insets) -> {
    Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
    v.setPadding(systemBars.left + mainPaddingDp,
                systemBars.top + mainPaddingDp,
                systemBars.right + mainPaddingDp,
                systemBars.bottom + mainPaddingDp);
    return insets;
});
```

Ovaj dio kôda osigurava da naš `ConstraintLayout` pravilno obrađuje sistemske prozore (kao što su statusna traka i navigacijska traka) dodavanjem odgovarajućeg paddinga.

Nakon što pokrenemo aplikaciju, vidjet ćemo da su `EditText` i `Button` pravilno postavljeni unutar `ConstraintLayout`, baš kao što je definirano u XML datoteci.

Sada ćemo dodati funkcionalnost gumbu tako da prikaže unos iz `EditText`-a u `Toast` poruci kada se klikne.

```
button.setOnClickListener(v -> {
    String inputText = editText.getText().toString();
    Toast.makeText(this, inputText, Toast.LENGTH_SHORT).show();
});
```

Moguće je dodati ovaj kôd odmah nakon što smo definirali `Button` i prije nego što primijenimo `ConstraintSet`, čak i ako nije dodan u `mainLayout`.

Programska izmjena prikaza

Pored programskog definiranja prikaza, moguće je i dinamički mijenjati postojeći prikaz tijekom izvođenja aplikacije. Na primjer, proširit ćemo našu prethodnu aplikaciju tako da kada kliknemo na gumb, tekst unesen u `EditText` bude prikazan u `TextView` koji ćemo dodati dinamički u `LinearLayout`. Također ćemo dodati dugme za brisanje svih unosa.

```
// Stvaramo reset button i postavljamo njegove atribute
Button buttonReset = new Button(this);

buttonReset.setText("Resetiraj");
buttonReset.getBackground().setTint(Color.parseColor("#FF9800"));
buttonReset.setId(Button.generateViewId());

buttonReset.setLayoutParams(new ViewGroup.LayoutParams(
    0, ViewGroup.LayoutParams.WRAP_CONTENT)
);
mainLayout.addView(buttonReset);

// Stvaramo linear layout i postavljamo njegove atribute
LinearLayout linearLayout = new LinearLayout(this);

linearLayout.setOrientation(LinearLayout.VERTICAL);
linearLayout.setPadding(mainPaddingDp/2, mainPaddingDp/2, mainPaddingDp/2,
    mainPaddingDp/2);
linearLayout.setBackgroundColor(Color.parseColor("#E6E6E6"));
linearLayout.setId(LinearLayout.generateViewId());

linearLayout.setLayoutParams( new LinearLayout.LayoutParams(
    0, 0)
);
mainLayout.addView(linearLayout);
...

// Constraint za Reset Button
set.connect(buttonReset.getId(), ConstraintSet.START, ConstraintSet.PARENT_ID,
    ConstraintSet.START);
set.connect(buttonReset.getId(), ConstraintSet.END, ConstraintSet.PARENT_ID,
    ConstraintSet.END);
// Constraint za Linear Layout
set.connect(linearLayout.getId(), ConstraintSet.START, ConstraintSet.PARENT_ID,
    ConstraintSet.START);
set.connect(linearLayout.getId(), ConstraintSet.END, ConstraintSet.PARENT_ID,
    ConstraintSet.END);

set.createVerticalChain(
    ConstraintSet.PARENT_ID, ConstraintSet.TOP,
    ConstraintSet.PARENT_ID, ConstraintSet.BOTTOM,
    new int[]{editText.getId(), buttonAdd.getId(), buttonReset.getId(),
        linearLayout.getId()},
    null,
    ConstraintSet.CHAIN_PACKED);
...
```

Potrebito je prethodno ažurirati vertikalni lanac kako bi uključio nove elemente.

Sada ćemo ažurirati `OnClickListener` za glavni gumb kako bismo dodali novi `TextView` u `LinearLayout` svaki put kada se klikne. Također ćemo dodati `OnClickListener` za reset gumb koji će ukloniti sve `TextView` elemente iz `LinearLayout`.

```
buttonAdd.setOnClickListener(v -> {
    String inputText = editText.getText().toString();
    TextView textView = new TextView(this);
    textView.setTextSize(20);
    textView.setText("- " + inputText);
    linearLayout.addView(textView);
});

buttonReset.setOnClickListener(v -> {
    linearLayout.removeAllViews();
});
```

Sada, svaki put kada korisnik unese tekst u `EditText` i klikne na gumb "Klikni me", novi `TextView` će biti dodan u `LinearLayout` s unesenim tekstrom. Kada korisnik klikne na gumb "Resetiraj", svi unosi će biti uklonjeni iz `LinearLayout`.

Dinamicko definiranje prikaza

mlijeko

DODAJ

RESETIRAJ

- jabuka
- banana
- kruh
- jaja
- mlijeko

+

-

1:1



Dinamičko dodavanje elemenata

Definiranje vlastitog prikaza

Pored korištenja ugrađenih Android prikaza, možemo kreirati i vlastite prilagođene prikaze koji nasljeđuju postojeće klase ili implementiraju vlastita sučelja.

Koristeći postojeće prikaze

Možemo se vratiti na prijašnji zadatak FoodTracker gdje imamo više `MainActivity`, `SettingsActivity` i `AddItemActivity`.



FoodTracker

Ovdje koristimo `TextView` widget za prikaz liste stavki. Umjesto da svaki put ažuriramo `TextView` koji nas dosta limitira na izgled, napraviti ćemo vlastiti **prikaz** stavke.

Prije toga ćemo promijeniti `TextView` widget u vertikalni `LinearLayout`, te isto tako ažurirati `MainActivity` da koristi linear layout i zakomentirati prijašnji kôd koji nam sada ne radi.

Sada možemo stvoriti novu klasu `StavkaView` i proširit je s `LinearLayout`, zatim ćemo unutar nje dodati dva `TextView` elementa za **naziv** i **kalorije** koje ćemo stvoriti i namjestiti u konstruktoru.

```
public class StavkaView extends LinearLayout {  
    private TextView textNaziv;  
    private TextView textKalorije;  
  
    public StavkaView(Context context, String naziv, float kalorije) {  
        super(context);  
        this.setOrientation(LinearLayout.HORIZONTAL);  
  
        textNaziv = new TextView(context);  
        textNaziv.setText(naziv);  
        addView(textNaziv);  
  
        textKalorije = new TextView(context);  
        textKalorije.setText(kalorije + " kcal");  
        addView(textKalorije);  
    }  
}
```

Kada pokrenemo imamo ovakav izgled:



StavkaView

Kada imamo vlastiti prikaz, onda ga možemo preuređiti kako želimo. Primjer:

```
public StavkaView(Context context, String naziv, float kalorije) {  
    super(context);  
    this.setOrientation(LinearLayout.HORIZONTAL);  
  
    setPadding(64, 32, 64, 32);  
  
    textNaziv = new TextView(context);  
    textNaziv.setText(naziv);  
    textNaziv.setTextSize(16f);  
  
    LayoutParams nazivParams = new LayoutParams(0, LayoutParams.WRAP_CONTENT, 1.0f);  
    addView(textNaziv, nazivParams);  
  
    textKalorije = new TextView(context);  
    textKalorije.setText(kalorije + " kcal");  
    textKalorije.setTextSize(16f);  
    textKalorije.setTypeface(null, Typeface.BOLD);  
    LayoutParams kalorijeParams = new LayoutParams(LayoutParams.WRAP_CONTENT,  
    LayoutParams.WRAP_CONTENT);  
    addView(textKalorije, kalorijeParams);  
}
```

Jabuka	62.0 kcal
Lubenica	256.0 kcal
Kruh	600.0 kcal
Maslac	250.0 kcal

Uređen prikaz StavkaView

Samostalna izrada prikaza

Sada kada znamo kako napraviti vlastiti prikaz za stavku, pokušat ćemo izraditi vlastiti prikaz za **cilj** `ProgressView` potpuno iz nule. Za početak, u `activity_main.xml` dodajemo jedan **ConstraintLayout** u koji ćemo smjestiti naš prilagođeni prikaz.

Prvo ćemo inicijalizirati nekoliko ključnih varijabli koje će nam služiti za praćenje stanja i crtanje prikaza:

- `float currentProgress & maxProgress`
 - Početna i maksimalna vrijednost napretka
- `Paint backgroundPaint, progressPaint & textPaint`
 - **Paint** objekt za crtanje

Također ćemo dodati pomoćnu metodu `initPaints()` za namještanje `Paint` objekata i **settere** za `currentProgress` i `maxProgress`:

- `Paint initPaints()`
 - Inicijalizacija i konfigura `Paint` objekt koji se koristiti kod `onDraw` metode
 - Omogućuje definiranje boja, debljine linija, stila ispune i ostalih grafičkih svojstava
- `setCurrentProgress(float currentProgress)`
- `setMaxProgress(float maxProgress)`

`invalidate()` metoda se koristi za **osvježavanje prikaza**, ponovo poziva `onDraw` i crta prikaz s ažuriranim vrijednostima. Obično se koristi nakon promjene stanja prikaza, u ovom slučaju kada se ažurira `currentProgress` ili `maxProgress`.

Budući da kreiramo vlastiti prikaz, moramo napraviti **@Override** za dvije ključne metode:

- `onMeasure(int widthMeasureSpec, int heightMeasureSpec)`
 - Ova metoda služi za određivanje dimenzija našeg prikaza unutar roditeljskog layouta
 - Budući da se element nalazi unutar **ConstraintLayout-a**, potrebno je pravilno postavljanje širine i visine, kao i osiguravanje da se prikaz skalira i reagira na ograničenja roditelja
- `onDraw(@NonNull Canvas canvas)`
 - Ovdje definiramo kako će naš prikaz izgledati
 - Koristeći `Canvas`, možemo crtati oblike, linije, tekst i boje koje predstavljaju napredak ili stanje cilja

`ProgressView` izgleda kao sljedeće:

```
public class ProgressView extends View {  
    private float currentProgress = 0;  
    private float maxProgress = 100;  
  
    private final Paint backgroundPaint;  
    private final Paint progressPaint;  
    private final Paint textPaint;  
  
    public ProgressView(Context context) {  
        super(context);  
        backgroundPaint = initPaints();  
        progressPaint = new Paint(backgroundPaint);  
        textPaint = new Paint(backgroundPaint);  
    }  
  
    @Override  
    protected void onMeasure(int widthMeasureSpec, int heightMeasureSpec) {  
        int desiredHeight = (int) TypedValue.applyDimension(  
            TypedValue.COMPLEX_UNIT_DIP, 32, getResources().getDisplayMetrics()  
        );  
        setMeasuredDimension(widthMeasureSpec, desiredHeight);  
    }  
  
    private Paint initPaints() {  
        Paint base = new Paint();  
        base.setAntiAlias(true);  
        base.setStrokeWidth(4);  
        base.setDither(true);  
        base.setFilterBitmap(true);  
        base.setStyle(Paint.Style.FILL);  
        base.setTextSize(16);  
        return base;  
    }  
}
```

```
base.setStyle(Paint.Style.FILL);
base.setAntiAlias(true);
return base;
}

protected void onDraw(@NonNull Canvas canvas) {
    super.onDraw(canvas);
    int width = getWidth();
    int height = getHeight();
    Log.d("TAG", "onDraw: " + height);

    // Pozadina
    backgroundPaint.setColor(Color.LTGRAY);
    canvas.drawRect(0, 0, width, height, backgroundPaint);

    // Progress
    float ratio = (maxProgress > 0) ? Math.min(currentProgress / maxProgress, 1f)
: 0f;
    int progressWidth = (int) (width * ratio);

    // Boja
    progressPaint.setColor(ratio < 1.0 ? Color.rgb(76, 175, 80) : Color.RED);
    canvas.drawRect(0, 0, progressWidth, height, progressPaint);

    // Tekst
    textPaint.setColor(Color.BLACK);
    textPaint.setTextSize(height / 2f);
    textPaint.setTextAlign(Paint.Align.CENTER);
    String text = currentProgress + " / " + maxProgress ;
    canvas.drawText(text, width / 2f, height / 2f + textPaint.getTextSize() / 2,
textPaint);
}

public void setCurrentProgress(float currentProgress) {
    this.currentProgress = currentProgress;
    invalidate();
}

public void setMaxProgress(float maxProgress) {
    this.maxProgress = maxProgress;
    invalidate();
}

}
```

Ukupno kalorija: 1168/2000.0

Jabuka	62.0 kcal
Lubenica	256.0 kcal
Kruh	600.0 kcal
Maslac	250.0 kcal

1168.0 / 2000.0

ProgressView