SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

mRanger

**PROJEKT U SKLOPU stem revolucija u zajednici**

Varaždin, 2018.

Sadržaj

[1. Uvod 1](#_Toc532922249)

[2. Instaliranje potrebnih programa 1](#_Toc532922250)

[3. Izrada prvog projekta u Android Studio 2](#_Toc532922251)

[3.1 Izrada novog projekta 2](#_Toc532922252)

[3.2 Prvi koraci u Android Studio alatu 6](#_Toc532922253)

[3.3 Pokretanje projekta 15](#_Toc532922254)

[4. Dodavanje novog zaslona, slike i gumba 18](#_Toc532922255)

[4.1 Dodavanje novog zaslona 18](#_Toc532922256)

[4.2 Dodavanje slike 20](#_Toc532922257)

[4.3 Dodavanje gumba 25](#_Toc532922258)

[4.4 Prijelaz s jednog zaslona na drugi 27](#_Toc532922259)

[4.5 Kreiranje layout-a 30](#_Toc532922260)

[5. Objekti i klase 33](#_Toc532922261)

[5.1 Dodavanje nove klase u Android Studio-u 34](#_Toc532922262)

[6. Izrada mRanger aplikacije 36](#_Toc532922263)

[6.1 Kreiranje potrebnih datoteka 37](#_Toc532922264)

[6.2 Dodavanje koda u klase 38](#_Toc532922265)

[6.3 Dodavanje koda u activity datoteke 42](#_Toc532922266)

[6.3 Dodavanje koda u layout datoteke 44](#_Toc532922267)

[6.4 Još par sitnih izmjena 47](#_Toc532922268)

[6.5 Sinkronizacija Gradlea 49](#_Toc532922269)

[7. Logičko programiranje 51](#_Toc532922270)

[7.1 Uvod u Prolog 51](#_Toc532922271)

[7.1.1 Tipovi podataka 51](#_Toc532922272)

[7.1.2 Činjenice i pravila 53](#_Toc532922273)

[7.1.3 Pravila 54](#_Toc532922274)

[7.2 Pisanje i izvršavanje Prolog programa 55](#_Toc532922275)

[7.3 Provjera godišnjeg doba 57](#_Toc532922276)

[7.4 Provjera prosječne temperature 58](#_Toc532922277)

[7.5 Pozivanje Prolog programa iz PHP-a 60](#_Toc532922278)

[7.6 Usporedba PHP i Prolog rješenja 62](#_Toc532922279)

[7.6.1 Provjera godišnjeg doba u PHP-u 62](#_Toc532922280)

[7.6.1 Provjera prosječne temperature u PHP-u 64](#_Toc532922281)

[7.7 Dodatni Prolog materijali 65](#_Toc532922282)

[7.7.1 Liste 65](#_Toc532922283)

[7.7.2 Rekurzije 65](#_Toc532922284)

[7.7.3 Posebni ugrađeni predikati 69](#_Toc532922285)

# Uvod

Cilj ovog dokumenta je upoznati mladež, čime su ciljana skupina učenici srednjih škola, kao i svi oni koji žele naučiti nešto više o robotici. U ovom dokumentu ćemo se potruditi što jednostavnije proći kroz proces izrade Android aplikacije koja će služiti za daljinsko upravljanje mBot Ranger robotom, na koji ćemo se povezati putem Bluetooth-a.

Kako bi što lakše pratili ovaj dokument, bez obzira na to što ćemo veliki dio gradiva mi sami objasniti, podrazumijevamo da imate minimalna predznanja o tome što su klase, objekti i funkcije … ili bar da ćete se potruditi to malo proučiti prije izrade ove aplikacije **korak po korak.** 😉

Aplikacija će imati sljedeće funkcionalnosti:

* Povezivanje na mBot Ranger putem Bluetooth-a
* Daljinsko upravljanje robotom
* Mogućnost podešavanja brzine kretanja
* Izbjegavanje sudara robota i prepreke
* Bilježenje temperature zraka u bazu podataka

# Instaliranje potrebnih programa

Našu aplikaciju ćemo izraditi u Android Studiu, a to je razvojno okruženje koje služi za kreiranje Android mobilnih aplikacija. Detaljni prikaz instalacije ćemo preskočiti, no postoje razni sadržaji na internetu koji prikazuju kako instalirati Android Studio (npr. [Instalacija Android Studia](https://www.youtube.com/watch?v=YaLAAdh9RBg)).

Android Studio se može preuzeti na: [Android Studio - Službena stranica](https://developer.android.com/studio/install)

# Izrada prvog projekta u Android Studio

U nastavku, sve upute ćemo se potruditi prikazati uz slikovne materijale i komentare kako bi što lakše predočili postupak izrade aplikacije. Na pojedinim slikama se mogu pronaći obojeni okviri i strelice kako bismo usmjerili vašu pozornost na bitne elemente.

Želimo Vam puno sreće prilikom izrade vaše prve aplikacije za kontroliranje metalnog ljubimca. 😉

## 3.1 Izrada novog projekta

Slika 1. Početni prozor Android Studio

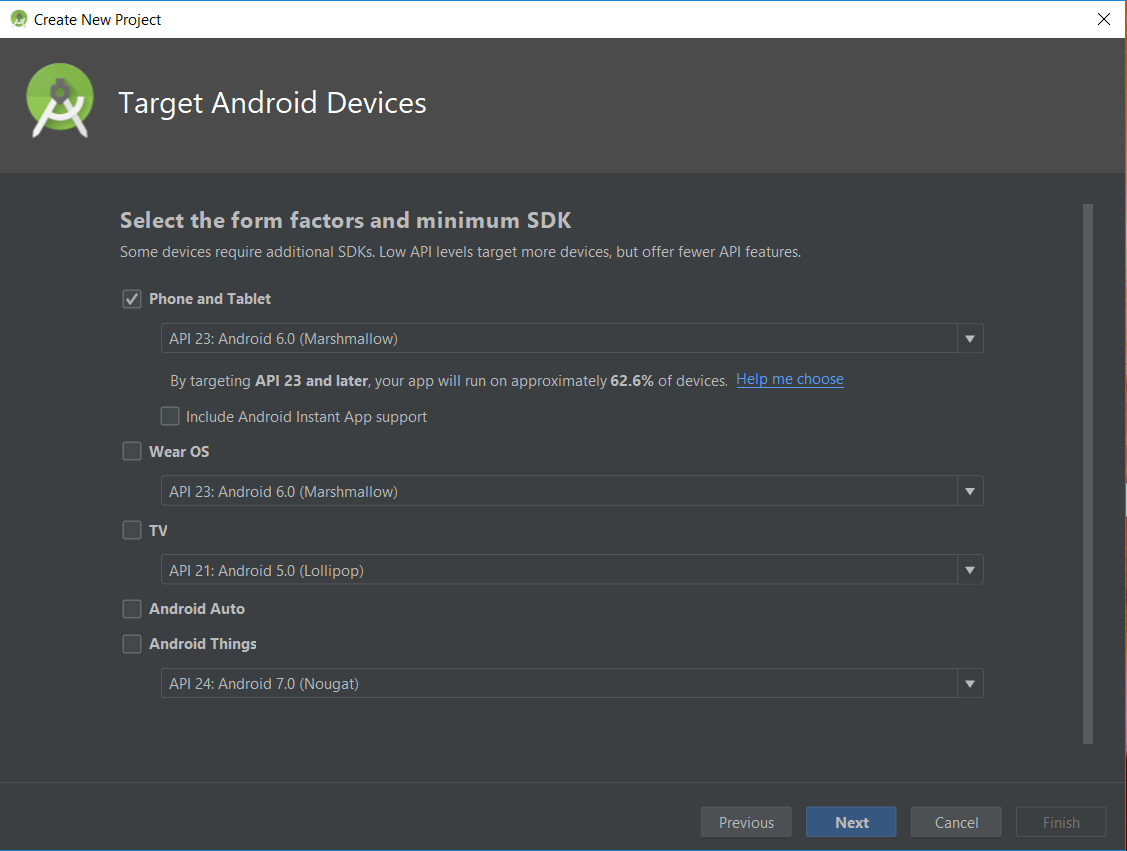
Prilikom pokretanja Android Studia otvara se prozor prikazan na slici iznad. Odaberemo prvu opciju iz prozora, *Start a new Android Studio project*, kojom ćemo započeti kreiranje novog projekta.

Time nam se otvara sljedeći prozor



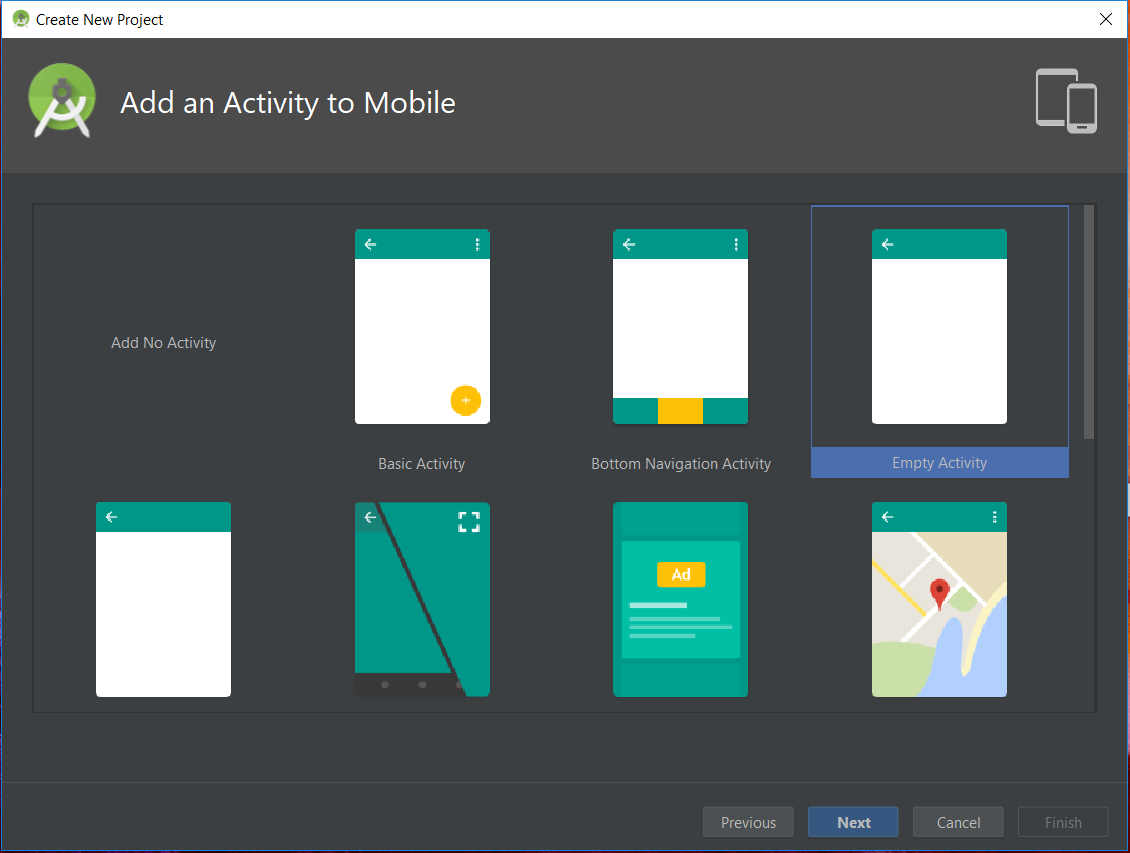
Slika 2. Imenovanje projekta

U ovom prozoru cilj je odrediti naziv projekta, podatke o tvrtki ako ona postoji (projekt će biti smješten u mapu *C:\Users\imeKorisnika\AndroidStudioProjects\tvrtka\mRanger).* Ostale opcije ostavljate takve kakve jesu i kliknete na *Next*.



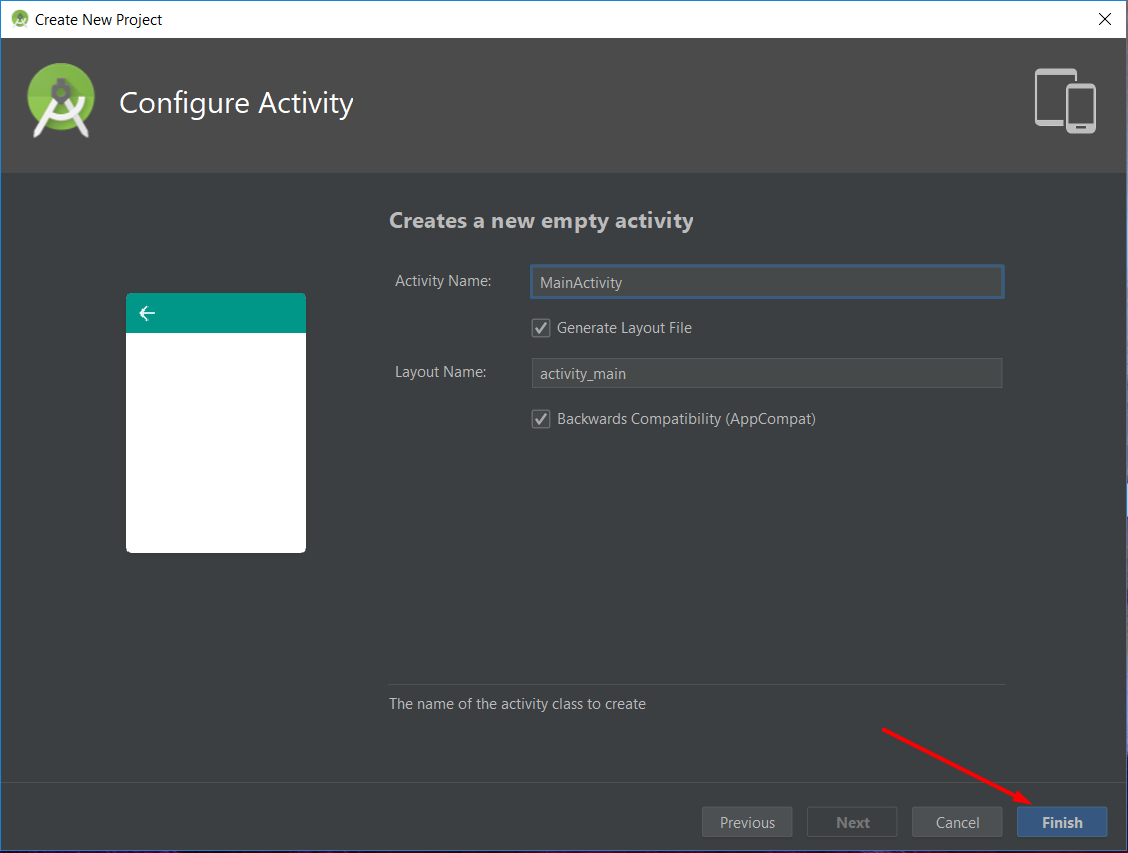
Slika 3. Odabir ciljanih uređaja

Nakon imenovanja projekta potrebno je odabrati vrste uređaje na kojima će se aplikacija izvoditi. U našem slučaju odabiremo *Phone and Tablet* opciju, te nakon toga biramo minimalnu verziju operacijskog sustava za android uređaje (minimalni zahtjevi sustava). Mi odabiremo predloženu opciju *API 23:Android 6.0 (Marshmallow)* prilikom čega dobivamo informaciju o postotku uređaja na kojemu će biti moguće koristiti aplikaciju. Zatim kliknemo *Next*.



Slika 4. Odabir početnog zaslona

U ovom prozoru imamo mogućnost jednog od predložaka za izgled početnog zaslona aplikacije. Odabiremo predloženi zaslon *Empty Activity* i kliknemo *Next*.



Slika 5. Naziv početnog zaslona

Na slici 5. je moguće odabrati željeno ime za početni zaslon, međutim ostavljamo sve kako je zadano i kliknemo na *Finish*.

## Slika na kojoj se prikazuje snimka zaslona, elektronički, monitor, zaslon Opis je generiran uz vrlo visoku pouzdanost3.2 Prvi koraci u Android Studio alatu

Slika 6. Sučelje Android Studio alata nakon otvaranja projekta

Sljedeća slika prikazuje sučelje Android Studio alata nakon otvaranja našeg novog izrađenog projekta. Na zaslonu nam se po početnim postavkama prikazuju dva taba (*activity\_main.xml* i *MainActivity.java*).

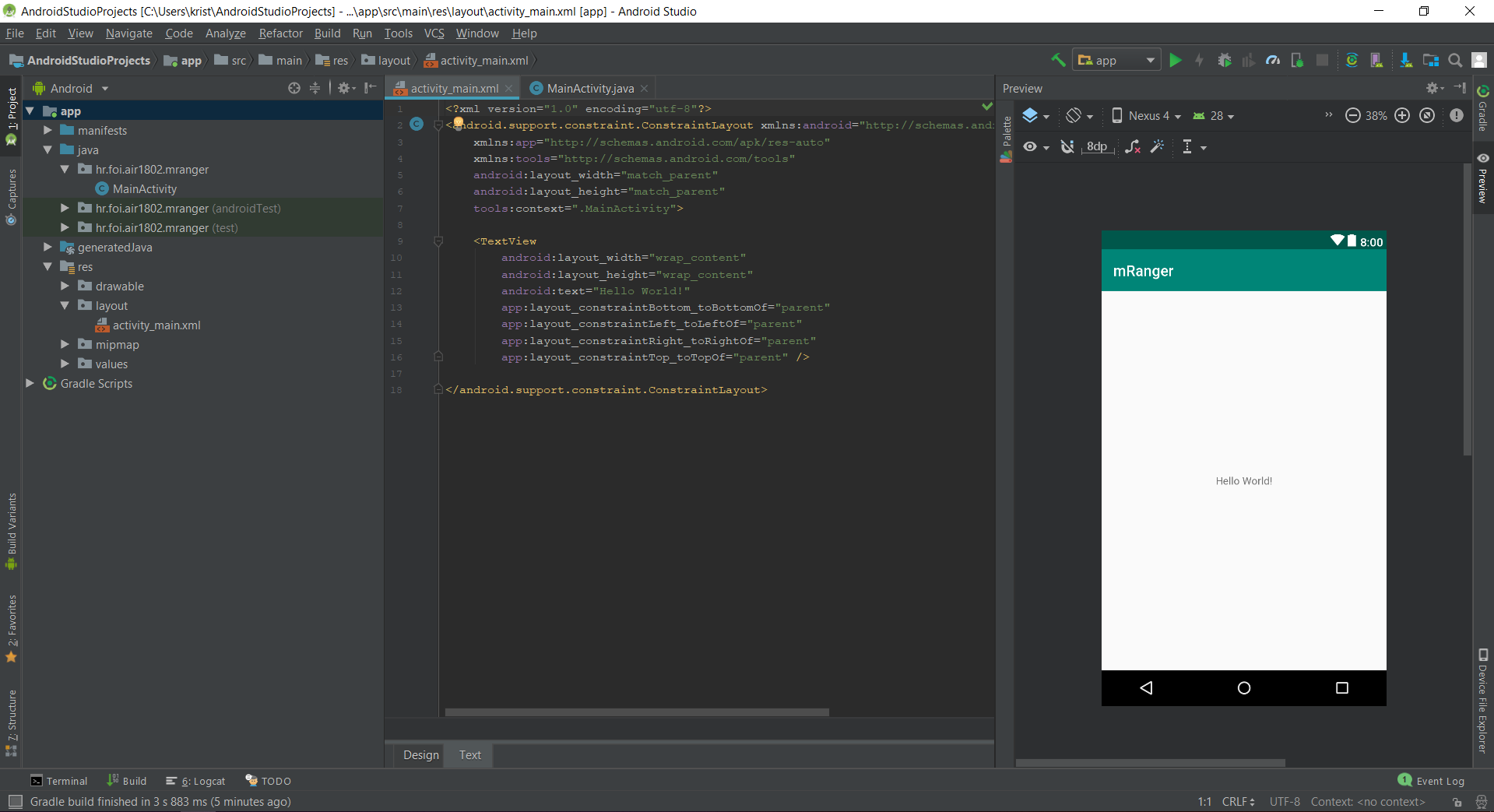
Slika na kojoj se prikazuje snimka zaslona, monitor, prijenosnik, sjedenje

Opis je generiran uz visoku pouzdanost

Na slici broj 7. vidimo početnu strukturu mape našeg projekta. Bitne mape su *java* i *res. U java* mapi, u podmapi sa nazivom naše tvrtke i aplikacije, nalazi se programski kod našeg početnog zaslona kojeg smo prethodno nazvali *MainActivity*. U mapi *res* u podmapi *Layout* nalazi se *activity\_main.xml*  u kojem se uređuje grafički izgled početnog zaslona naše aplikacije.

Slika 7. **java** i **res** mape

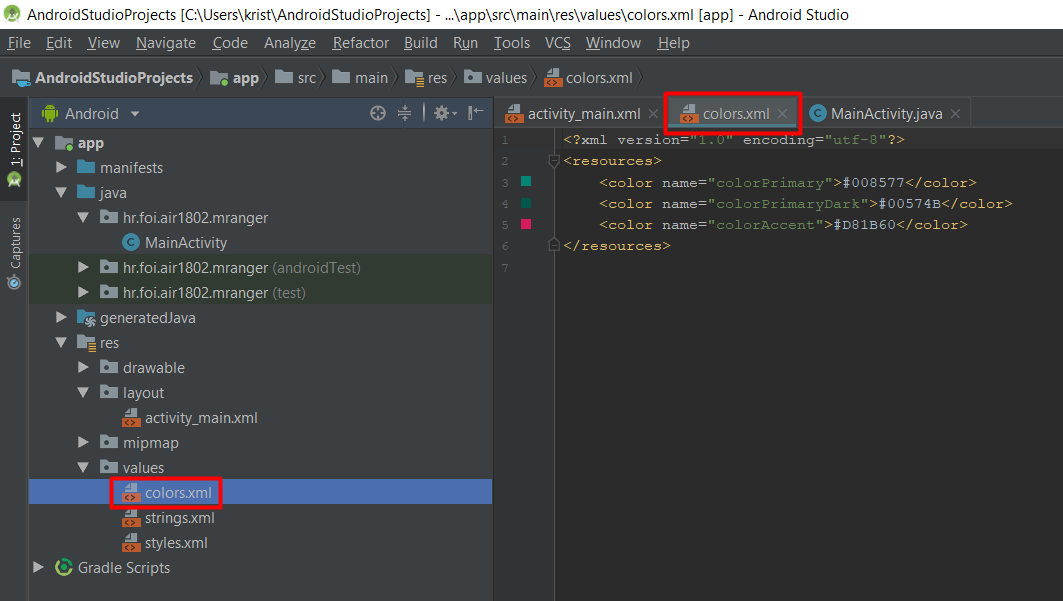
U dokumentu *MainActivity.java* nalazi se programski kod koji se izvršava na početnom zaslonu.



Slika 8. **XML** datoteka

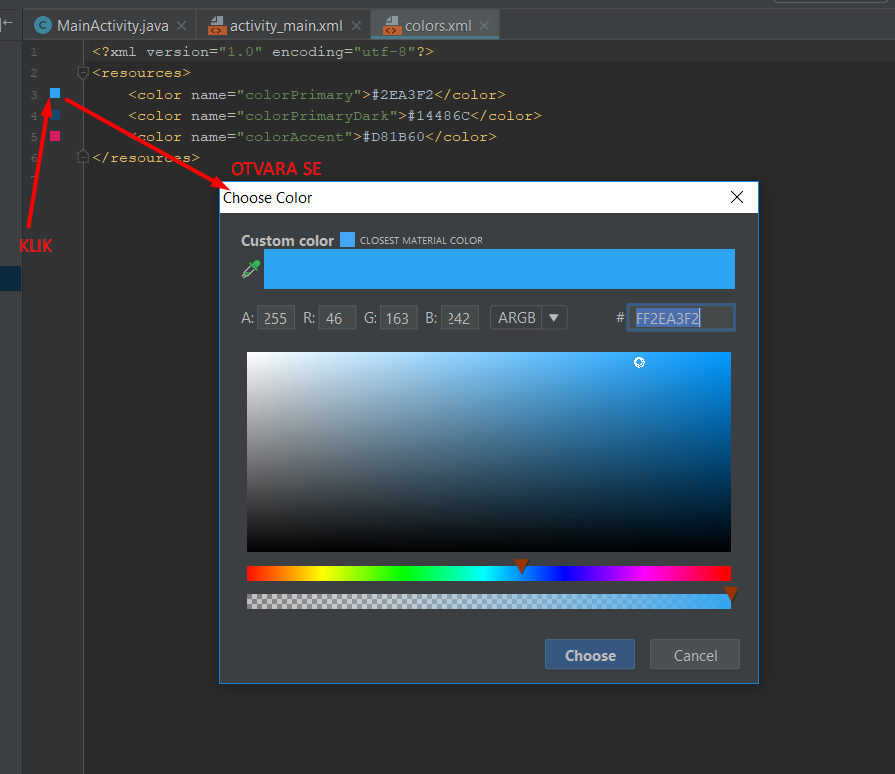
U dokumentu *activity\_main.xml* se nalazi programski kod u XML jeziku, koji definira grafički izgled našeg zaslona te podataka koje ćemo prikazivati na njemu. Po početnim postavkama Android Studio alata, na sredini zaslona se prikazuje tekst *Hello World!* Taj tekst možemo također uređivati u ovom XML dokumentu, no o tome ćemo reći nešto više ubrzo.

Predložene boje našeg početnog zaslona su nijanse zelene. Idemo to malo izmijeniti, ne želimo zelenu aplikaciju ! 😉

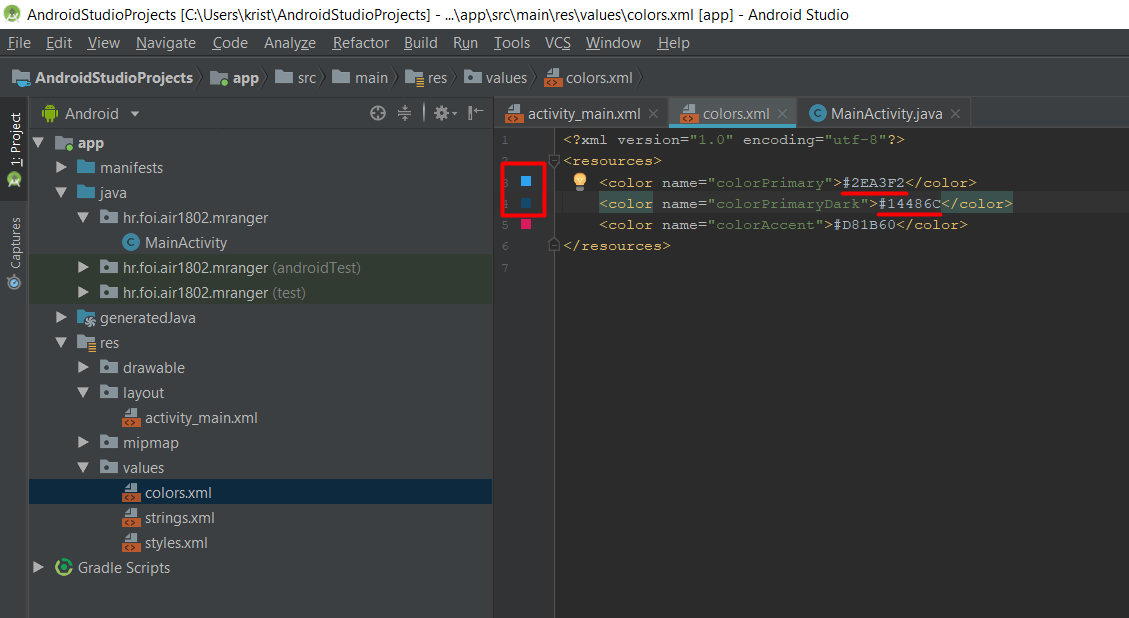


Slika 9. Promjena boja zaslona 1/2

U prethodno spomenutom **res** mapi, imamo mapu **values** koja sadrži datoteku **colors.xml** gdje možemo promijeniti boje zaslona naše aplikacije. Kao što možete vidjeti, boje zaslona su zapisane u heksadecimalnom zapisu.

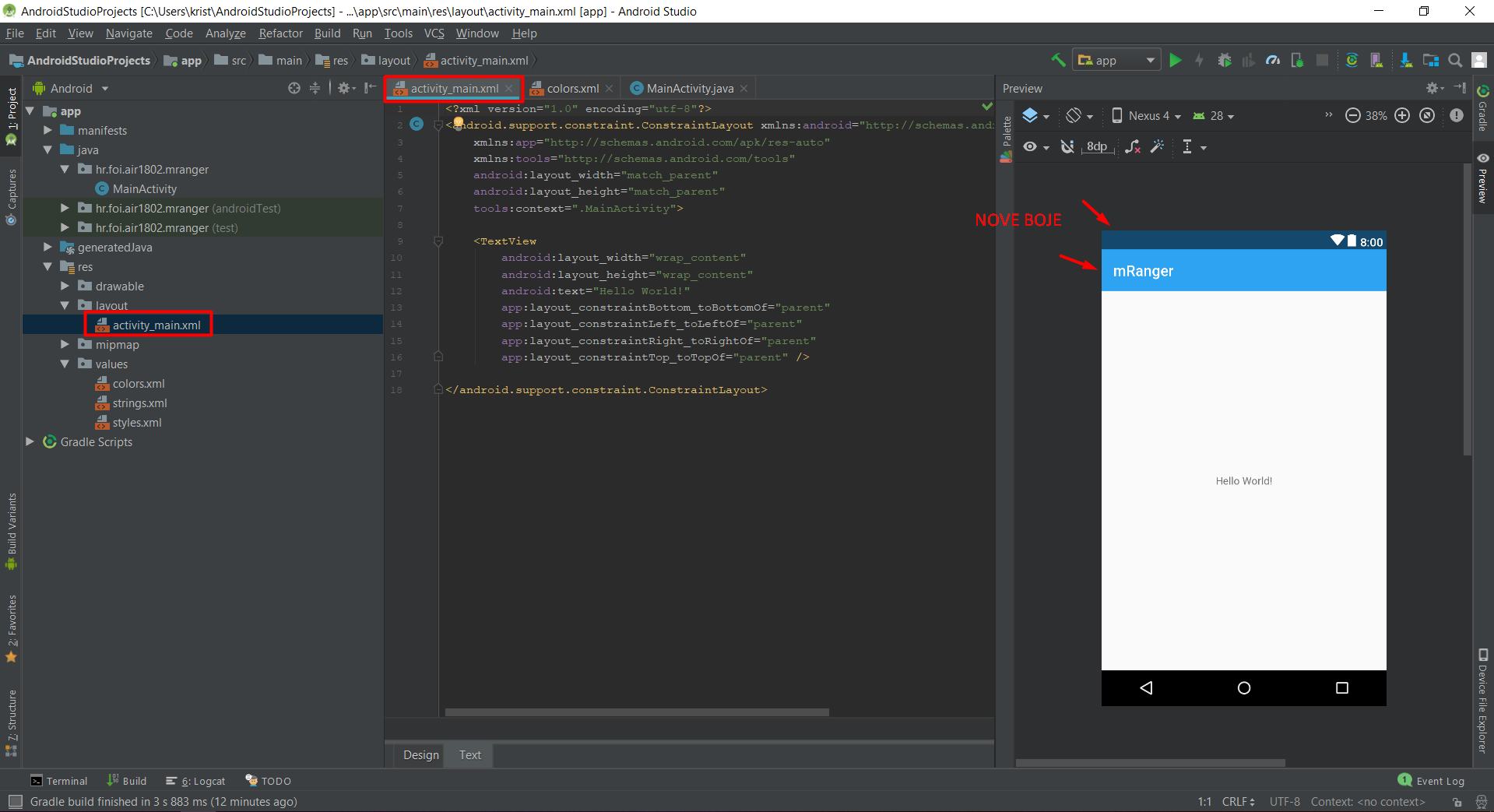
Klikom na kvadratić sa bojom pokraj linije koja sadrži heksadecimalni zapis boje, otvara nam se prozor gdje možemo odabrati željenu boju na vrlo jednostavan način, bez potrebe da znamo stvarni heksadecimalni kod boje.

Slika 10. Promjena boja zaslona 2/2



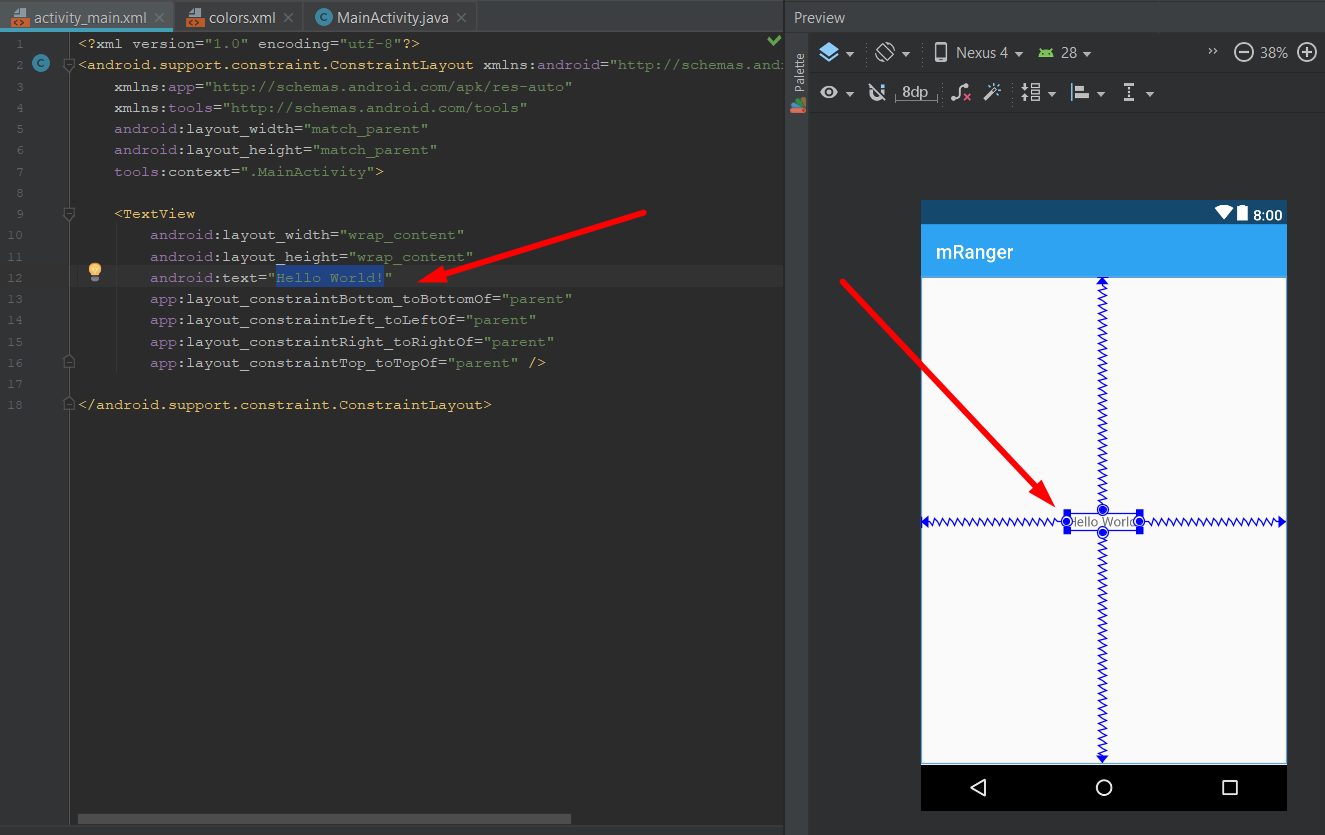
Slika 11. Promjena heksadecimalnog zapisa boje

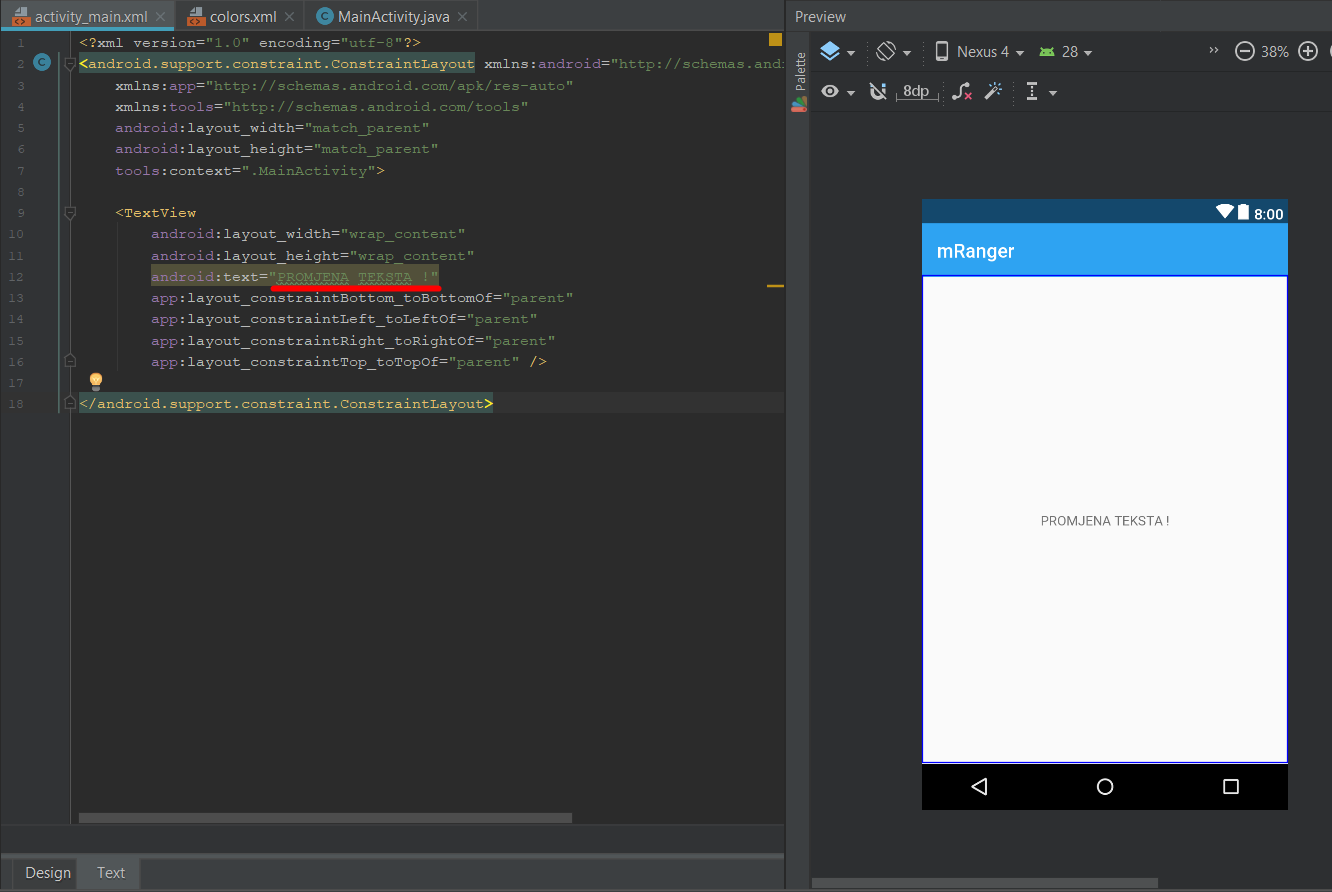
Nakon odabira željenih boja, možemo vidjeti kako se promijenio heksadecimalni kod u **colors.xml** dokumentu.

 Otvorimo li *activity\_main.xml,* možemo vidjeti prikaz našeg zaslona te uočiti da su se boje izmijenile.

Slika 12. Promjena boja zaslona

Također, idemo malo izmijeniti i ovaj tekst *Hello World!* Koji se sam od sebe stvorio u našoj aplikaciji prilikom kreiranja. Ako malo bolje promotrimo *activity\_main.xml* uočavamo da postoji redak gdje je taj tekst zapisan.





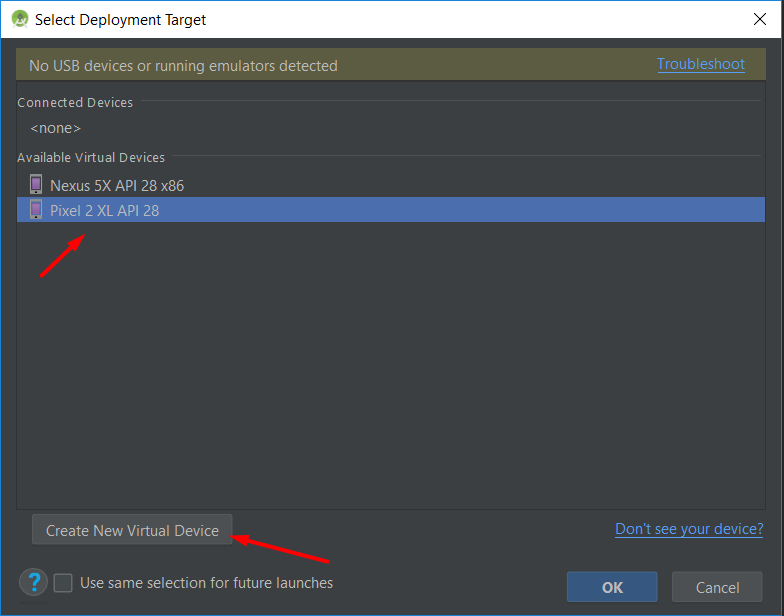
Slika 13. Tekst u XML-u

Slika 14. Promjena teksta

## 3.3 Pokretanje projekta

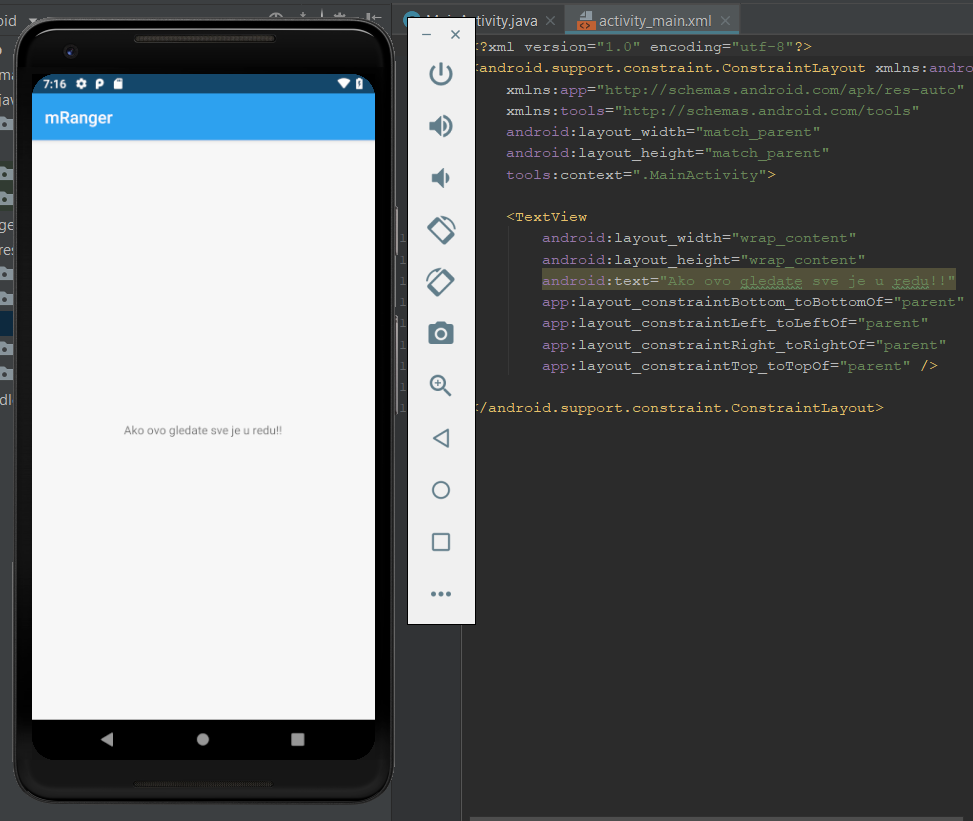
Slika 15. Gumb za pokretanje projekta

Kada poželimo vidjeti našu aplikaciju na djelu, u gornjem desnom dijelu alatne trake se nalazi gumb za pokretanje aplikacije.



Slika 16. Odabir virtualnog uređaja

Nakon toga prikazuje se zaslon za odabir virtualnog uređaja na kojem želimo pokrenuti aplikaciju. Možemo odabrati jedan od već ponuđenih ili pak dodati novi uređaj. Klik na *OK.*

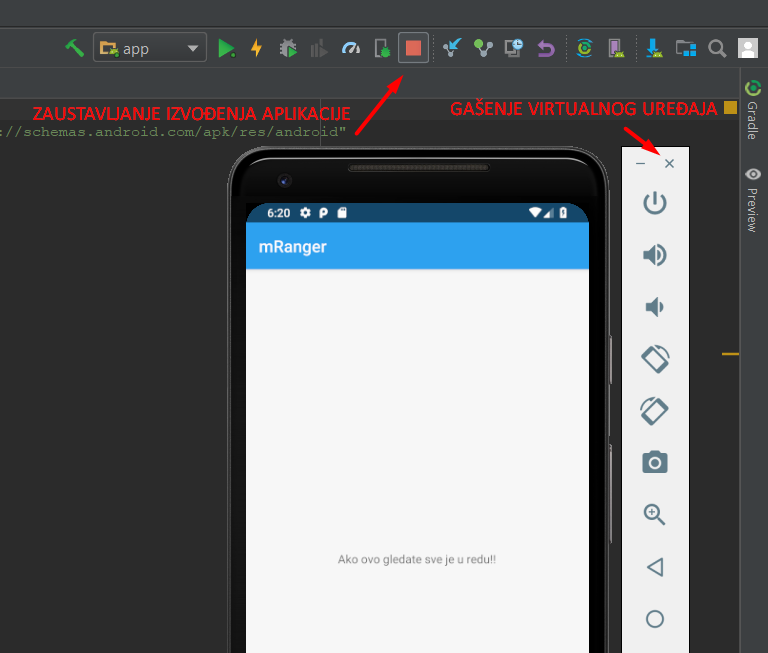
Prilikom pokretanja virtualnog uređaja automatski se pokrene i prikaže na zaslonu naša aplikacija.

Slika 17. Prikaz aplikacije na virtualnog uređaju uslijed pokretanja

Važno je napomenuti:

* Prilikom programiranja u Android Studio alatu, jednom kada pokrenemo virtualni uređaj nije potrebno gasiti, već samo minimiziramo dok radimo promjene u kodu.
* Ukoliko vaše računalo nema minimalno 8 GB RAM memorije, vrlo vjerojatno nećete moći koristiti značajku virtualnog uređaja, no tada imate opciju priključiti svoj pravi uređaj putem USB, te odabrati ga prilikom pokretanja kao na slici 16.
* Emulator nema savršeno ugrađen rad s Bluetooth-om te se preporučuje za rad s našom aplikacijom spajanje vlastitog Android uređaja.

Prilikom zaustavljanja izvođenja aplikacije (pogledati sliku 18. ispod) ne gasimo virtualni uređaj, već zaustavljamo samo izvođenje aplikacije pritiskom na *Stop* znak koji se pojavi nakon pokretanja u gorenjem desnom dijelu alatne trake.



Slika 18. Prekid izvođenja aplikacije

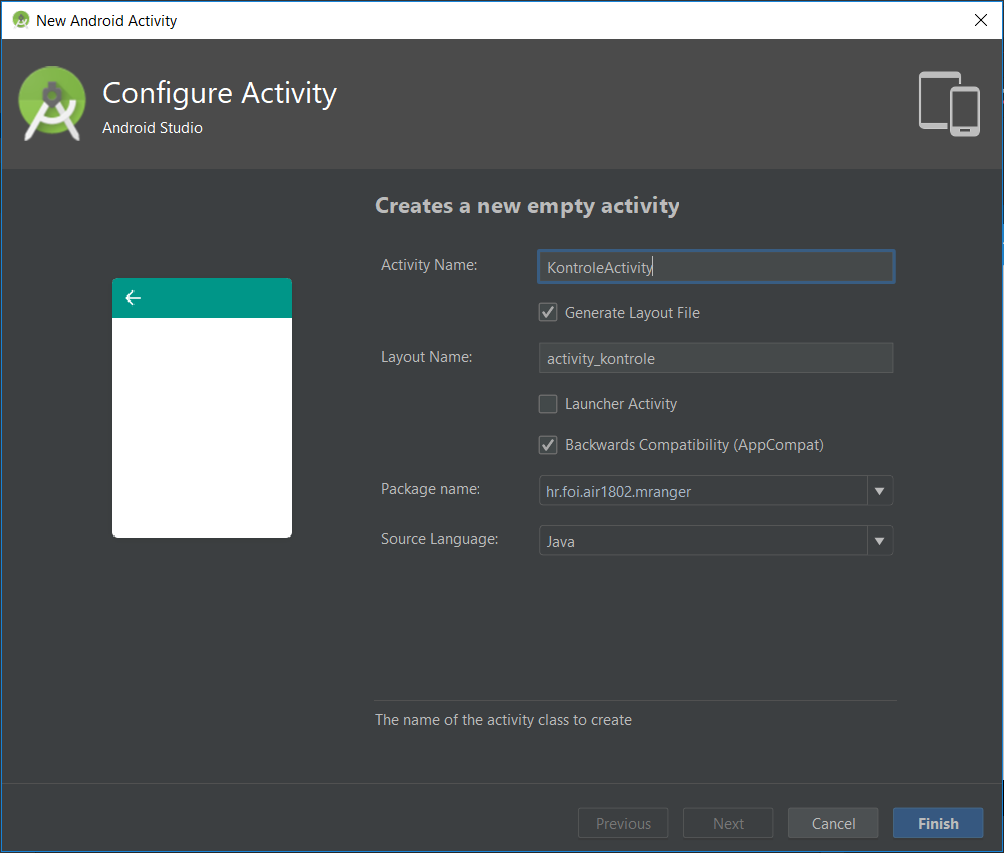
# Dodavanje novog zaslona, slike i gumba

U prethodnom poglavlju stvorili smo i uredili početni zaslon sa pripadajućim datotekama, *MainActivity.java* i *activity\_main.xml*. Pošto se najprije moramo spojiti Bluetooth-om na robota, taj početni zaslon će nam služiti za spajanje mobilnog uređaja na robot putem Bluetooth-a. Isto tako nakon spajanja, želimo imati zaslon putem kojeg ćemo upravljati robotom, tako da ćemo sada dodati novi zaslon u naš projekt.

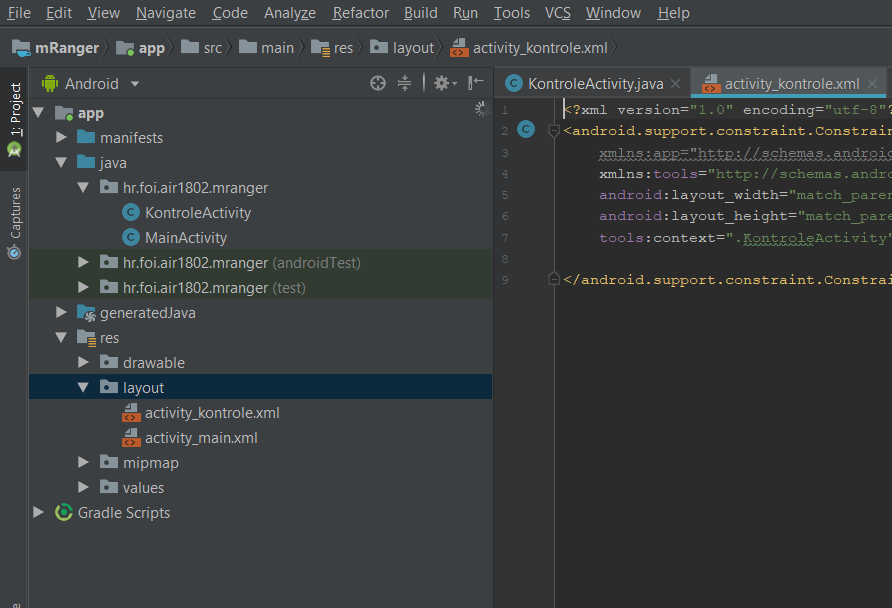
## 4.1 Dodavanje novog zaslona

Slika 19. Kreiranje novog zaslona (nove aktivnosti)

Slika 20. Imenovanje novog zaslona (nove aktivnosti)



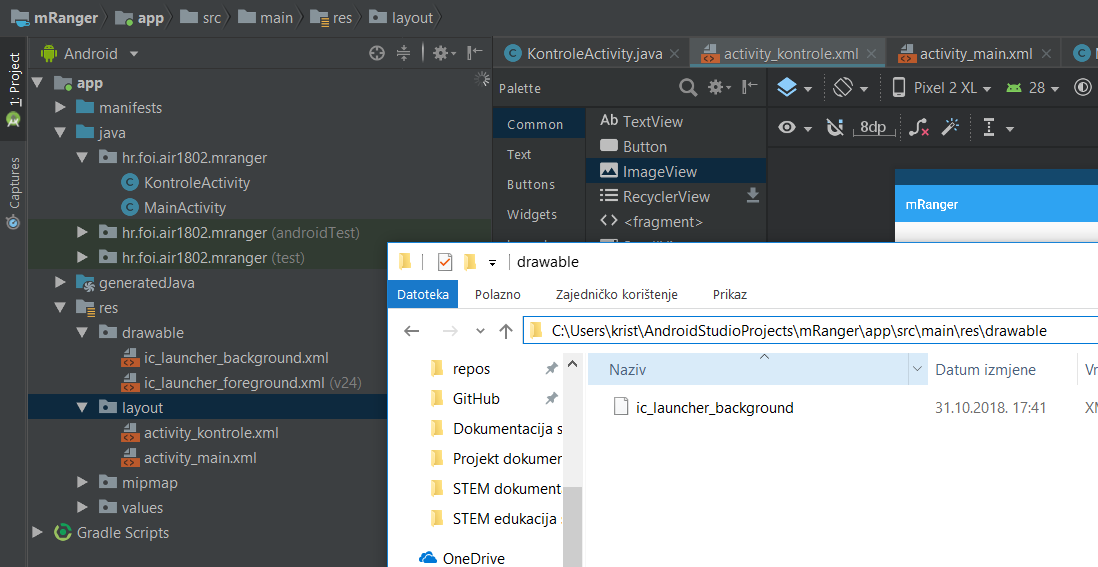
Novokreirani zaslon će sadržavati kontrole (joystick) za upravljanje mBot uređajem te ga stoga nazivamo *KontroleActivity*. Razlog takvome nazivu je nekakva praksa da programeri u androidu različite zaslone smatraju i nazivaju aktivnostima.

 Uz svaki zaslon dolaze dvoje datoteke, jedna za *layout* u kojoj se uređuje izgled. Druga je *Java klasa* u kojoj se nalazi pripadajući kod koji će se izvršavati za taj zaslon.

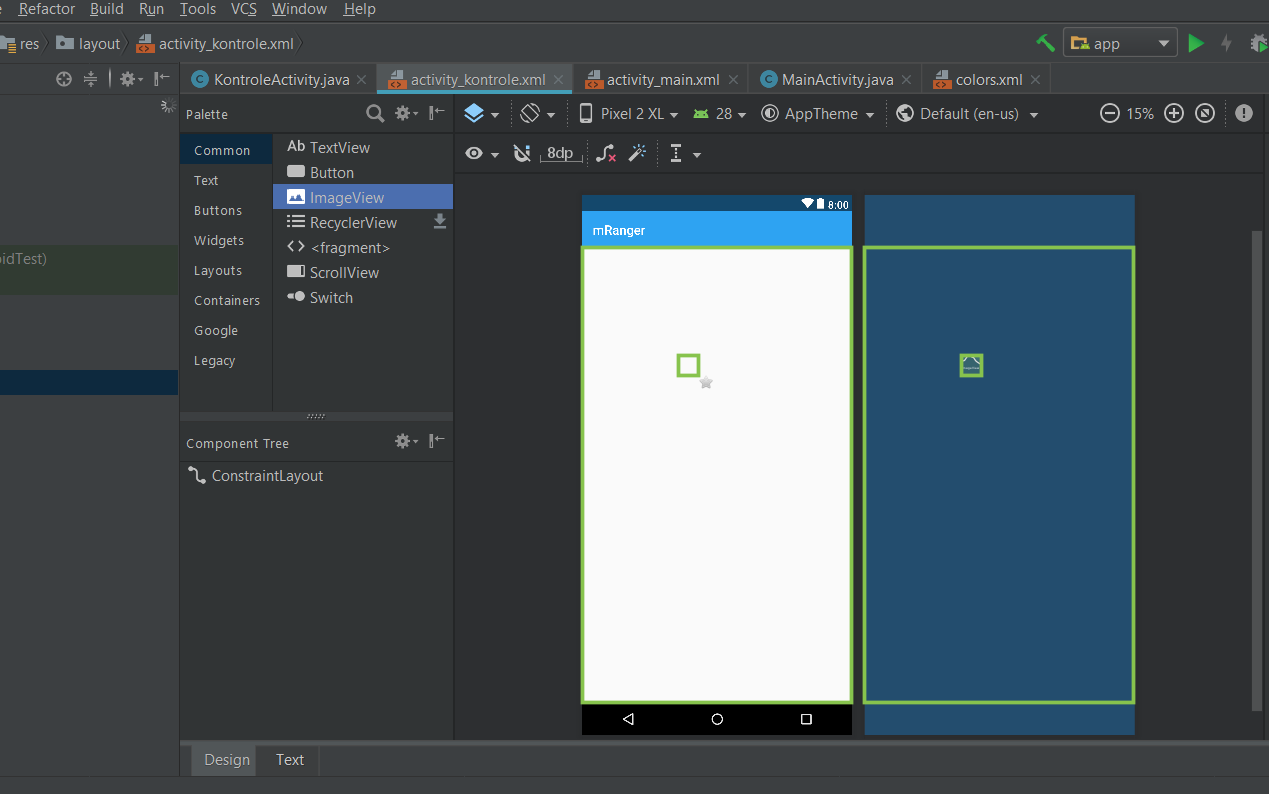
Slika 21. Stvoren novi zaslon

## 4.2 Dodavanje slike

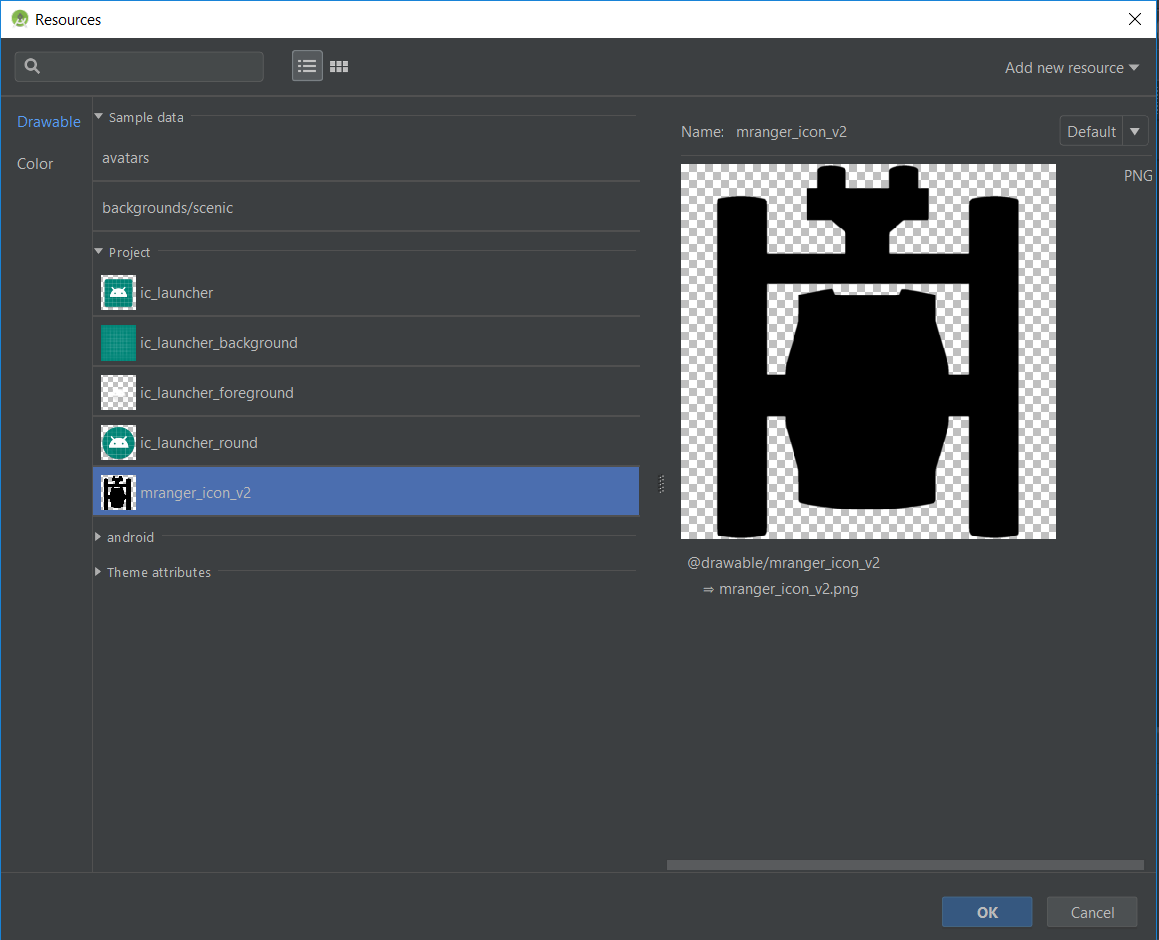
U sjedećem poglavlju pokazati ćemo kako se slika dodaje na zaslon, te koji su sve koraci potrebi za to. Kako bi razlikovali ovaj novi zaslon, na njega ćemo staviti sliku koju ćemo poslije iskoristiti. Kako bi slike dodavali na zaslon u Android Studio-u, te slike moraju biti u posebnom projektnoj mapi.

 Na slici se također nalazi putanja do spomenute mape (mapa *drawable*) u koju prije dodavanja slike je potrebno postaviti željenu sliku. Ime slike mora obavezno biti napisano malim slovima, te ako se radi o više riječi, one moraju biti odvojene donjom crtom („\_“).

Slika 22. Dodavanje slike u mapu projekta

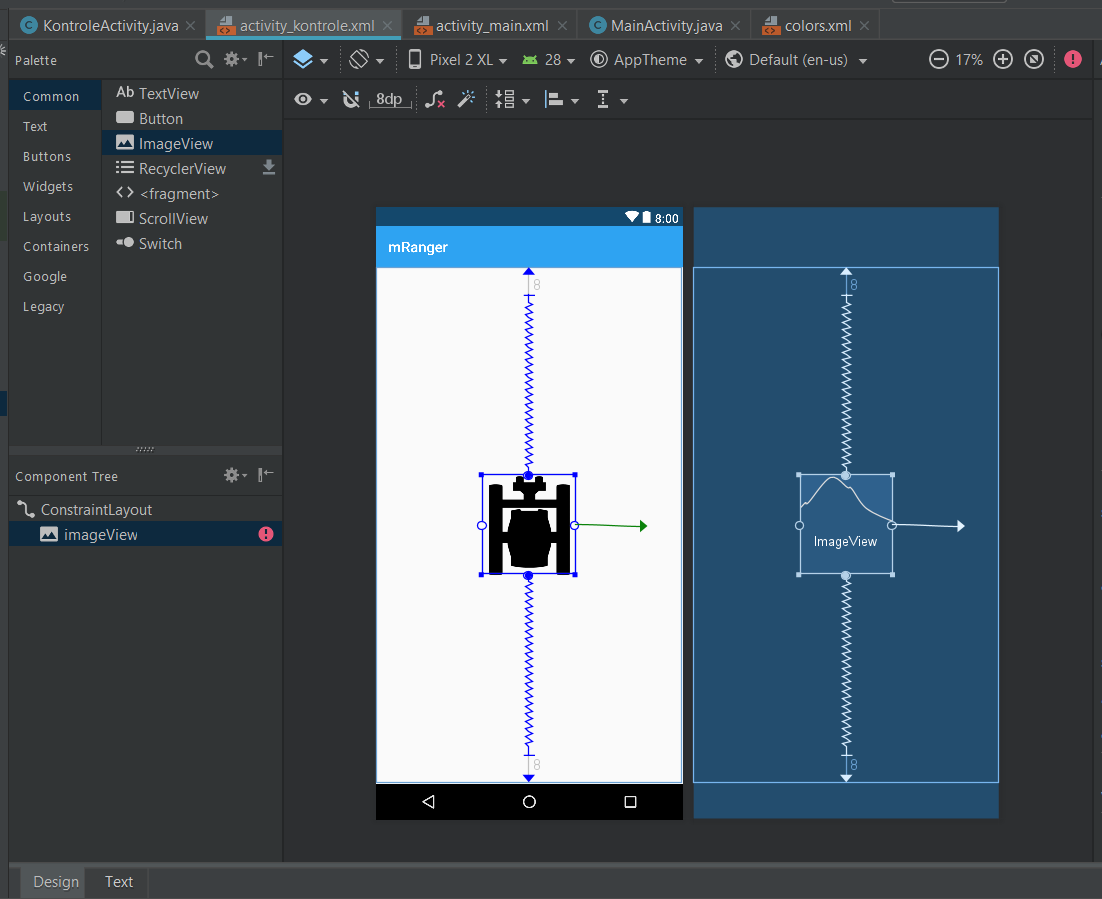


Slika 23. Dodavanje slike na zaslon

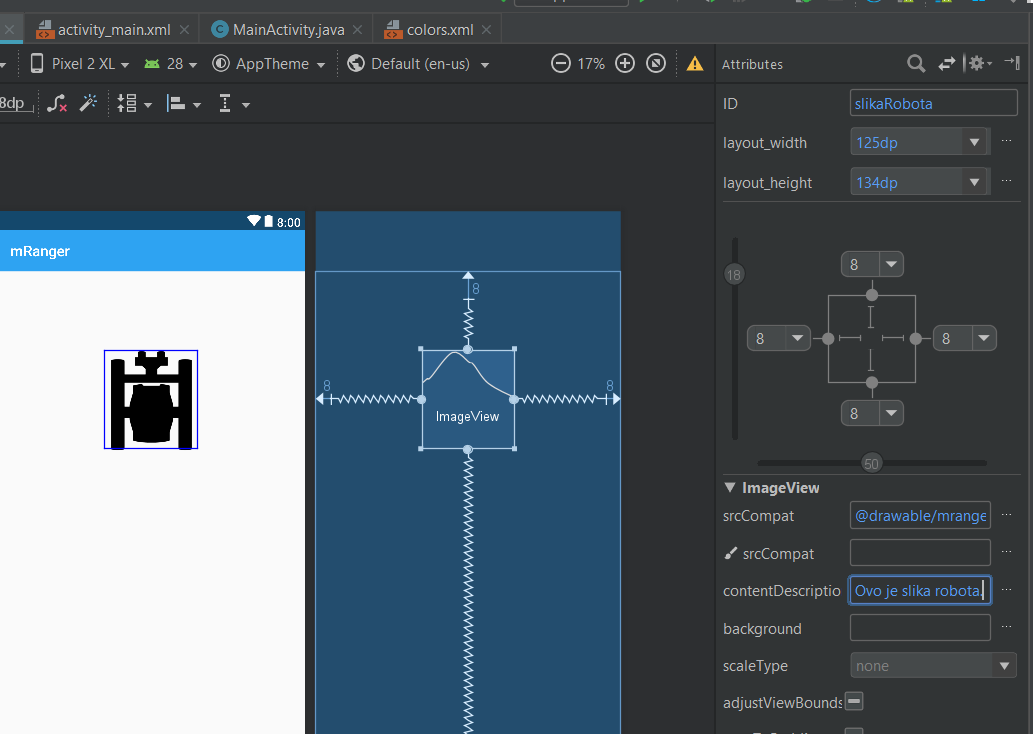
 Kako bi dodali sliku na zaslon, unutar *Design* taba dodajemo kontrolu *ImageView* koja predstavlja okvir unutar koje će se slika prikazivati. Kontrola se na zaslon dodaje po principu povuci i spusti (*eng. drag and drop*).

Slika 24. Odabir slike koja će se prikazat

Nakon ispuštanja kontrole, otvara nam se sljedeći prozor gdje biramo sliku koja će se prikazati u okviru. Tu ćemo na odabir imati neke predložene slike od Android Studio-a, kao i one koje smo dodali u *drawable* mapu.

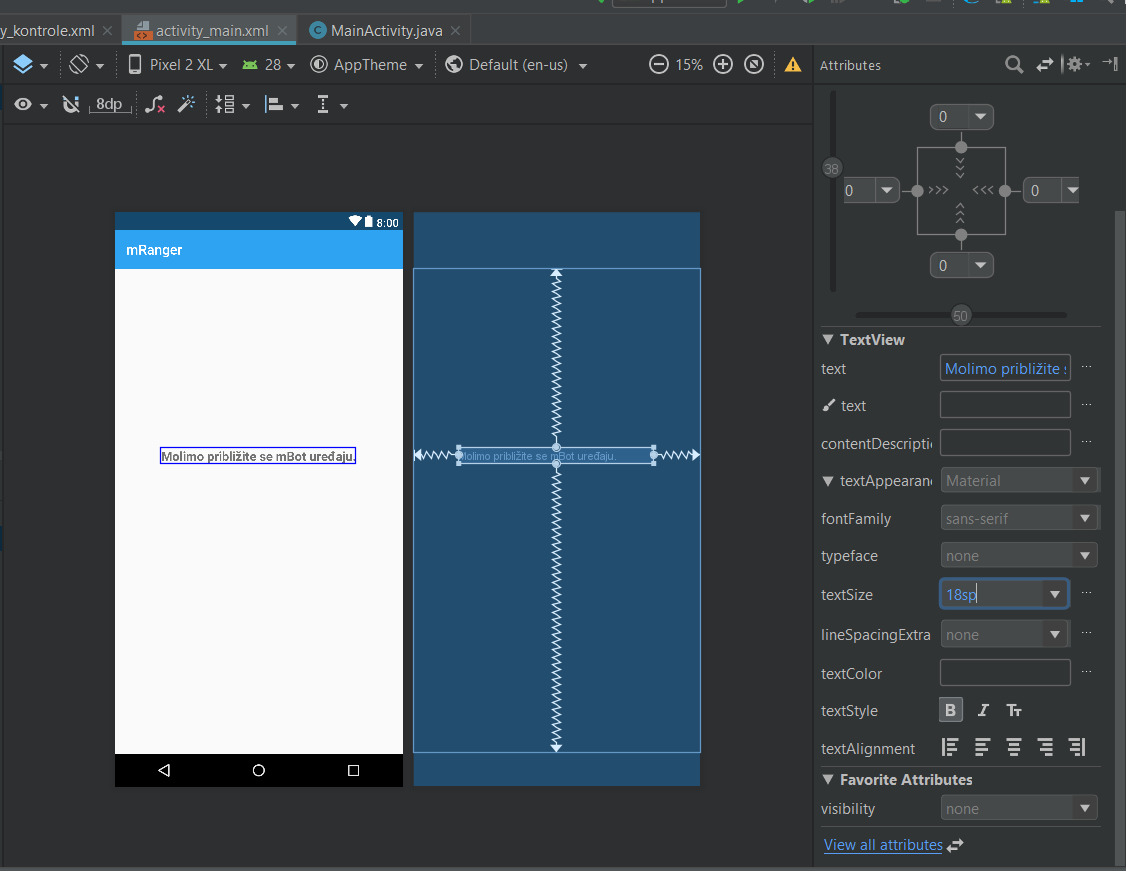
Kako bi se naša slika nalazila na željenom mjestu, a moramo uzeti u obzir da postoje zasloni različitih veličina, potrebno je sliku „usidriti“ sa svih njezinih strana okvira. Sliku usidravamo povlačenjem crte od točke sa sredine njezinog okvira pa sve do krajnjih rubova zaslona i/ili drugih kontrola.

Slika 25. Smještaj slike na zaslonu

 Novo dodanoj kontroli koja sadrži sliku dodajemo *id* kako bi je mogliprepoznati, te dajemo opis i ostale atribute po želji (proučite).

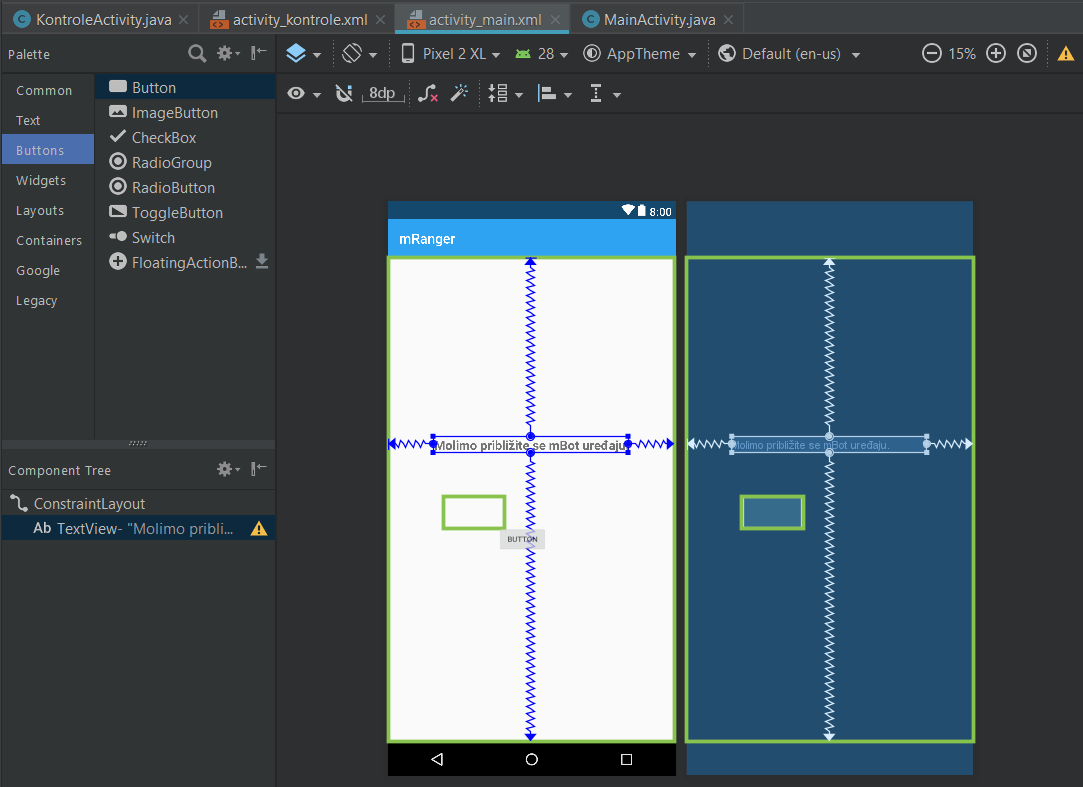
Slika 26. Dodavanje id-a i opisa slike

## 4.3 Dodavanje gumba



Slika 27. Sidrenje tekstualnog okvira

Iz istog razloga kao i kod slike sidrimo kontrolu tekstualnog okvira na željenu poziciju na zaslonu. Pošto je ovo naš početni zaslon, on se prvi otvara i na njemu se nalazi aktivnost povezivanja robota putem Bluetooth-a. Kako bi započeli proces povezivanja, dodajemo gumb pomoću kojeg ćemo to izvršiti.



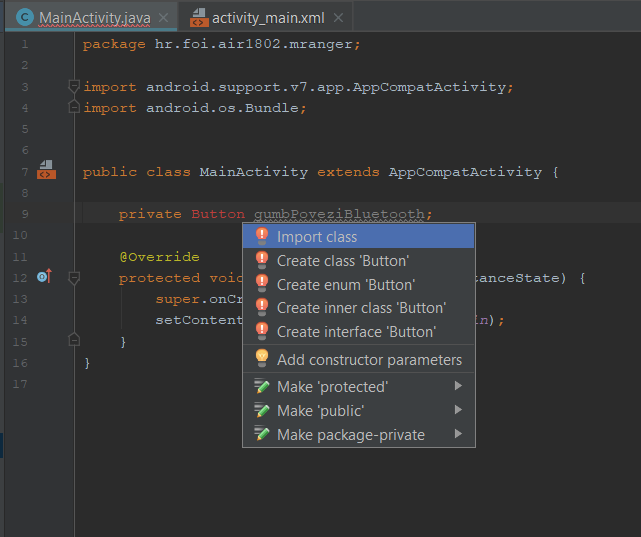
Slika 28. Dodavanje gumba

Istim principu kako smo dodali kontrolu za slikovni okvir, dodajemo kontrolu gumba na zaslon. Sidrimo je isto kao i kod kontrole slikovnog okvira. Također dodajemo joj *id* (taj naziv *id*-a će poslije biti jako važan u kodu), kao i tekst koji će se prikazivati na njemu, te ostale atribute po želji.

## 4.4 Prijelaz s jednog zaslona na drugi

Slika 29. Kontrola za gumb

Pozicioniramo se unutar *MainActivity.java* datoteke, te u njoj pišemo pripadajući kod za prijelaz na zaslon *KontroleActivity.java.*

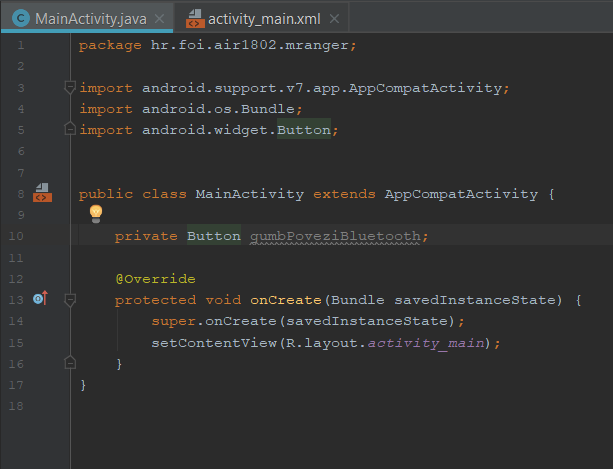


Slika 30. ALT + Enter kombinacija tipaka

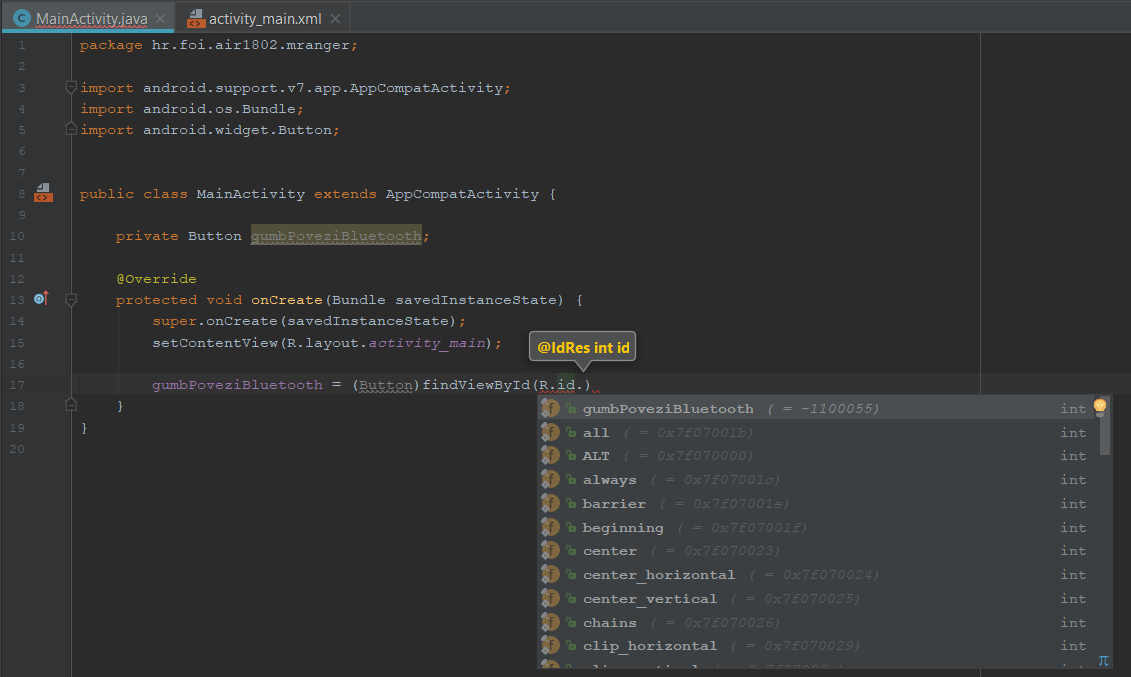
Napomena: ova kombinacija tipki je vrlo važna, ona omogućuje brzo izvršavanje unaprijed predviđenih opcija, te služi kao prva opcija koju ćemo poduzeti ukoliko dođe do nekih greški.

Na primjer, na slici poviše je napisana sljedeća linija koda gdje stvaramo novu privatnu varijablu tipa *Button* koju nazivamo po želji, ali zbog jednostavnosti isto onako kao što je i *id* našeg gumba.

Vidimo kako se pojavila greška kod tipa varijable jer nismo posebno uključili biblioteku koja se koristi za rad s tim tipom kontrole, te jednostavnim *Alt + Enter* dobivamo nekoliko mogućih opcija za rješavanje tog problema. U ovom slučaju je potrebno odabrati *Import Class* kako bi dodali biblioteku i otklonili pogrešku.

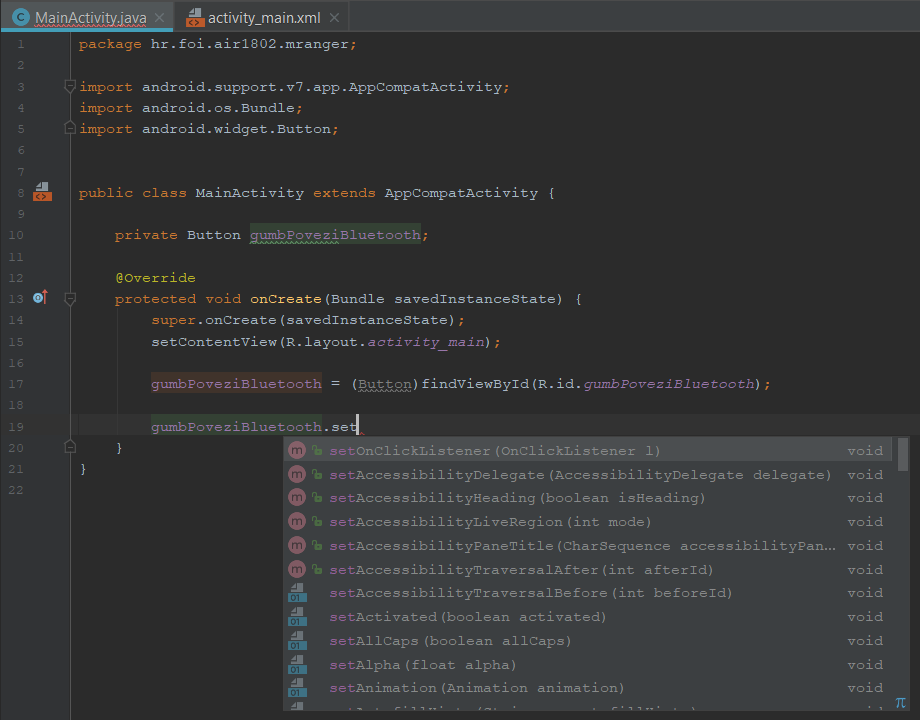


Slika 31. Dodana biblioteka

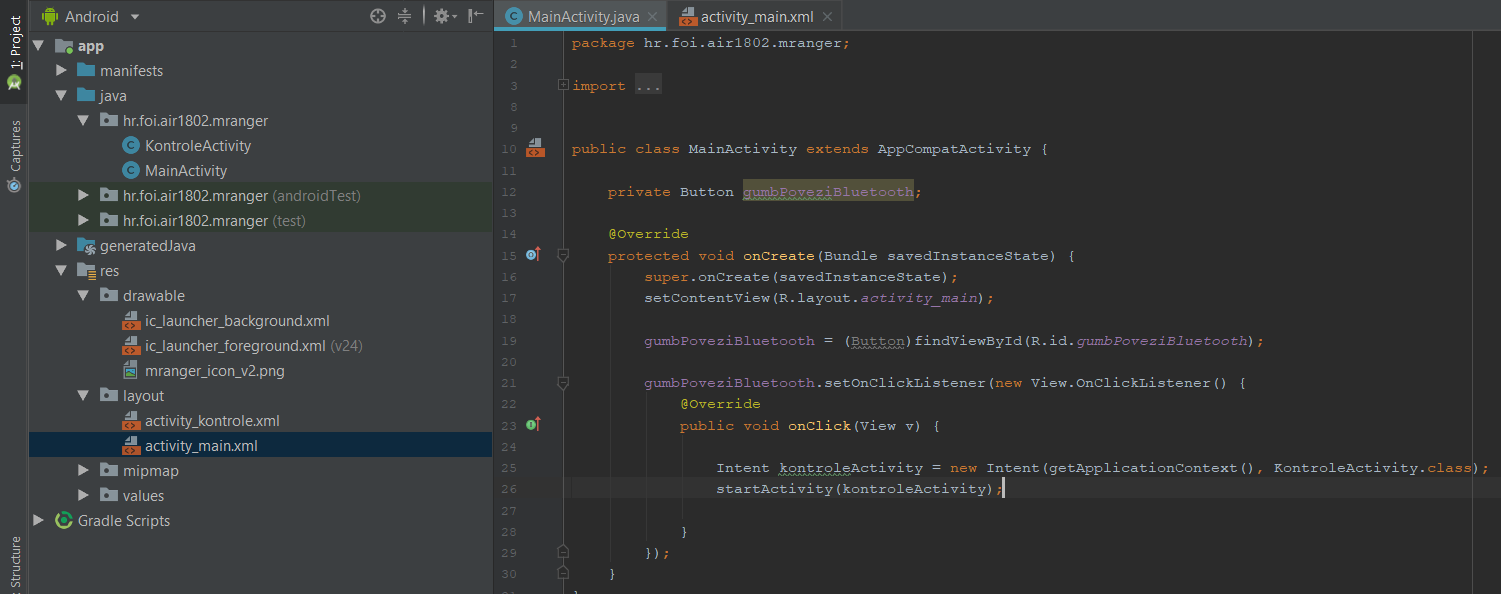


Slika 32. Dohvaćanje gumba preko id-a

Varijabli koju smo prethodno stvorili u kodu dajemo vrijednost, odnosno povezujem je sa kontrolom pripadajućeg gumba koji se nalazi na našem zaslonu. Primjetimo kako korištenje točke kod određenih elemenata nam također daje više mogučnosti što olakšava tipkanje koda kao i pronalaženje one prave.



Slika 33. Dodavanje setOnClickListener metode

Dodavanje *setOnClickListener* metode na gumb omogućujemo izvršavanje aktivnosti koje ćemo mi odrediti prilikom klika na određeni gumb na zaslonu.

Slika 34. Dodavanje koda koji se izvršava na klik gumba

Unutar onClick metode nalazi kod pomoću kojeg se prilikom pritiska na gumb prelazi sa početnog zaslona na željeni, što je u ovom slučaju *KontroleActivity.java.*

## 4.5 Kreiranje layout-a

Još jedna vještina koja će Vam biti potrebna za izradu ovog projekta je dodavanje nove *layout* datoteke. Kao što ste i dosad već mogli vidjeti, kreiranjem nekog novog *activity-a*, kreira se i njegova datoteka za izgled tipa *.xml,* koja se stavlja u **layout** mapu. Ta *layout* datoteka služi kako bi svojim kodom vizualno definirala izgled tog zaslona.

Međutim, ta *layout* datoteka se može i samostalno napraviti kako bi definirali izgled nekih drugih elemenata. Mi ćemo dalje u kodu definirati izgled *ListView* elementa (liste koja će prikazivati detektirane Bluetooth uređaje) te ćemo za njega morati izraditi posebnu *layout* datoteku.

Slika na kojoj se prikazuje snimka zaslona, monitor

Opis je automatski generiranSlika na kojoj se prikazuje monitor, snimka zaslona, zid, crno

Opis je automatski generiranDajemo datoteci željeno ime i dalje stisnemo OK. (Ne diramo ništa više 😉 )

Slika 36. Imenovanje layout resource datoteke

Slika 35. Dodavanje nove layout resource datoteke

Slika na kojoj se prikazuje snimka zaslona

Opis je automatski generiran

Slika 37. Dodana nova layout resource datoteka

Ovako bi trebao izgledati rezultat kreiranja nove *layout resource* datoteke, a kod koji će potrebno biti dodati, moći će te kopirati također iz ovog dokumenta kasnije kada dođemo do tog dijela.

# Objekti i klase

Sljedeća stvar koju Vam moramo objasniti prije nego li možemo nastaviti sa izradom aplikacije, su pojmovi **klasa, objekt** i **metoda.**

**Objekt** u realnom svijetu, može biti recimo: automobil ili bicikl, a u programskoj igri lopta ili neki lik.

Svaki objekt je definiran stanjem i ponašanjem; na primjer, stanje lika u igri mogu biti **atributi** poput: broj godina, ime, prezime, visina, težina itd., a ponašanje lika možemo definirati **metodama** poput : Trči, Skači, Spavaj. Ispod slijedi dio koda kojim ćemo to bolje objasniti.

(Dvostrukim klikom na kod možete ga kopirati)

**U programu je objekt opisan varijablama koje mu određuju stanje i metodama koje mu određuju ponašanje.**

**Klasa** predstavlja nacrt (predložak) objekta (eng. class). Stoga se uvijek prvo kreiraju klase na temelju kojih se proizvode objekti. Na primjer, kada kreiramo klasu za sportaša, u igri možemo kreirati više likova koji mogu imati drugačije atribute (različite godine, imena, visine …) i ponašanja (drugačije metode).

## 5.1 Dodavanje nove klase u Android Studio-u

Slika na kojoj se prikazuje snimka zaslona, monitor, crno, zid

Opis je automatski generiranU ovom odlomku ćemo Vam prikazati kako se kreiraju i dodaju nove klase u projekt. Na slikama možete popratiti pojedine korake.

Slika 39. Unosimo željeno ime i tip: Class

Slika 38. Dodavanje nove klase u projekt

Slika na kojoj se prikazuje snimka zaslona, monitor, crno, sjedenje

Opis je automatski generiran

Slika na kojoj se prikazuje snimka zaslona, monitor

Opis je automatski generiranOd ovog trenutka, upoznati ste sa svim potrebnim predznanjima kako bi mogli kreirati ovu aplikaciju prateći naše daljnje upute, te svi koraci od sada pa na dalje se baziraju na ovim prethodnim objašnjenim.

Slika 40. Dodana je nova klasa u projekt

Naravno, sav kod ćemo Vam mi objasniti i priložiti kako bi ga Vi mogli zalijepiti na predviđena mjesta.

# Izrada mRanger aplikacije

Sada slijedi najbitniji dio ovog edukacijskog dokumenta, a to je zapravo izrada aplikacije mRanger. Kako smo svjesni da ste vi još uvijek novi u radu sa Android Studiom, te možda izradom aplikacije bilo kakve vrste uopće, ovo ćemo vam nastojati objasniti na najjednostavniji mogući način.

Prvo ćemo od Vas zatražiti da izradite potrebne *Activity-e* odnosno zaslone, potrebne **klase** te potrebne **resource layout** datoteke, a zatim ćemo Vam priložiti kod koji ćete moći kopirati u te datoteke kako bi aplikacija radila.

Sav kod, sve klase i metode koje smo izradili smo Vam objasnili te ćete moći shvatiti i zašto su oni tu gdje jesu, odnosno koja je njihova zadaća u ovoj cijeloj priči.

Kad smo sve to rekli, krenimo na posao. 😊

*POTREBNO JE PAŽJIVO PRATITI UPUTE !*

*SRETNO* ❤

## 6.1 Kreiranje potrebnih datoteka

Prije nego što počnete sa izradom ovog projekta, zatvorite trenutni projekt na kojem smo dosad radili i pokazivali osnove rada u Android Studio-u, te kreirajte novi projekt gdje ćete pod *Application name* staviti **mranger**, te pod *Company domain* staviti **air1802.foi.hr**, kako ne bih morali mijenjati niti jednu liniju prilikom kopiranja koda, jer smo ga i mi također nazvali na taj način.

Slika na kojoj se prikazuje snimka zaslona

Opis je automatski generiran(*Project location* možete odabrati po vašoj volji)

KREIRAJTE NOVE KLASE S IMENIMA KAO NA SLICI

(NE ACTIVITY, NEGO BAŠ KLASE !!)

DODATI NOVU LAYOUT RESOURCE DATOTEKU IMENA: discover

AUTOMATSKI ĆE SE DODATI SA ZASLONOM KontroleActivity

KREIRAJTE NOVI ZASLON (ACITVITY) IMENA : KontroleActivity

OVO SE KREIRA ZAJEDNO SA KREIRANJEM PRIJEKTA

OVO SE KREIRA ZAJEDNO SA KREIRANJEM PRIJEKTA

Slika 41. Popis datoteka projekta

Pobrinite se da izradite sve poviše navedene datoteke sa pravim imenom. Nakon što ih kreirate, ne morate se brinuti za njihov sadržaj, odnosno kod jer će te unutra sve izbrisati prije nego što kopirate naš.

## 6.2 Dodavanje koda u klase

Prethodno smo kreirali 4 klase : **Bluetooth, ConnectBT, Controls** i **DeviceListAdapter.** Sada ćemo u jednu po jednu od njih zalijepiti pripadajući kod.

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **Bluetooth** klasu, iz koje se prethodno tome sve obrisali.



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **ConnectBT** klasu, iz koje se prethodno tome sve obrisali.



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **Controls** klasu, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **DeviceListAdapter** klasu, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

## 6.3 Dodavanje koda u activity datoteke

Prethodno smo kreirali 2 activity-a : **MainActivity** i **KontroleActivity.**

Sada ćemo u jednu po jednu od njih zalijepiti pripadajući kod.

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **MainActivity** datoteku, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **KontroleActivity** datoteku, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

## 6.3 Dodavanje koda u layout datoteke

Prethodno smo kreirali 3 layout datoteke : **activity\_kontrole.xml, activity\_main.xml** i **discover.xml.**

Sada ćemo u jednu po jednu od njih zalijepiti pripadajući kod.

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **activity\_main.xml** datoteku, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

Prije nego što zalijepimo kod za **activity\_kontrole.xml**, na njoj se nalazi slika, te ćemo vam ispod priložiti sliku koju će te spremiti na vaše računalo, te zatim dodati u projekt kako je to opisano u poglavlju [**4.2 Dodavanje slike**](#_4.2_Dodavanje_slike)**.**

**Desni klik** na sliku te zatim **Spremi kao sliku…** te je nazovite **mranger\_icon\_v2** u već ponuđenom **PNG** formatu.

**Slika na kojoj se prikazuje tekst

Opis je automatski generiran**

Slika 42. Slika za dodat

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **activity\_kontorle.xml** datoteku, iz koje se prethodno tome sve obrisali.

  
**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **discover.xml** datoteku, iz koje se prethodno tome sve obrisali.

**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

## 6.4 Još par sitnih izmjena

Dodali smo kod u skoro sve potrebne datoteke. Naime još samo moramo izmijeniti kod na dva mjesta.

**Slika na kojoj se prikazuje snimka zaslona

Opis je automatski generiran**

Slika 43. Dodatne datoteke za uredit

UREĐIVATI ĆEMO OVE DATOTEKE

1. Sljedeći kod treba u cijelosti kopirati i zalijepiti u **AndroidManifest.xml** datoteku, iz koje se prethodno tome sve obrisali.



1. **Slika na kojoj se prikazuje snimka zaslona

   Opis je automatski generiran**Sljedeći kod treba **samo jednu liniju** kopirati i zalijepiti u **build.gradle (Module: app)** datoteku, iz koje se **prethodno tome niste ništa obrisali !!!**

Slika 44. Dodatna linija

Linija koda koju treba kopirati je:

**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

## 6.5 Sinkronizacija Gradlea

Slika na kojoj se prikazuje elektronički

Opis je automatski generiran Naposljetku, potrebno je još samo sinkronizirati *Gradle* i naša aplikacija je spremna za pokretanje.

SINKRONIZACIJA GRADLEA

Slika 45. Sinkronizacija projekta

*AKO STE SVE ISPRAVNO NAPRAVILI, VAŠA APLIKACIJA JE SPREMNA ZA POKRETANJE NA VAŠEM ANDROID UREĐAJU !*

***ČESTITAMO I HVALA VAM NA PAŽNJI !* ❤**

mRanger tim 🏎🌫

# Logičko programiranje

Sljedeći dio ovog edukacijskog dokumenta baviti će jednim novim poglavljem, s kojim se zasigurno dosad niste imali prilike susresti. Riječ je o jednom logičkom programskom jeziku, koji je znatno drugačiji u logici rada od ostalih programskih jezika koji se koriste danas. Riječ je o Prologu, jeziku pomoću kojeg se može ostvariti nešto što će rezultirati, ili bar izgledati, kao umjetna inteligencija.

Znate ono, roboti, sami donosu odluke, zaključuju i slično ? E pa tako nešto, naša je ideja u Prolog jeziku obraditi podatke poput temperature i vremena mjerenja s naše prethodno izrađene mRanger aplikacije, te ostvariti da Prolog donese neke zaključke poput koje je godišnje doba te da li je temperatura veća ili niža od one prosječne za to doba dana u godini na primjer.

U nastavku teksta ćemo pokazati osnove Prolog jezika, koji su njegovi sastavni elementi te kako se oni zajedno koriste.

**Prolog**

Prolog je programski jezik namijenjen za logičko programiranje, a pogotovo za razvoj umjetne inteligencije te od tuda njegov naziv, ***PRO****graming in* ***LOG****ic*.

## 7.1 Uvod u Prolog

*Prolog jezik se temelji na predikatnoj logici prvog reda i koristi ograničenu verziju klauzula oblika poznatu kao Hornorov klauzalni oblik.*

Što to znači ? Pa ništa posebno vama vjerojatno, stoga ćemo i u nastavku teksta koristiti neke manje *stručne* izraze, pa ako ikada budete više učili o Prologu, oprostite nam na jednostavnosti.

Prolog bi tako bio koristan za opisivanje *objekata* i *relacija* među njima, recimo na primjer osoba je *objekt*, a ako je osoba roditelj nekoj drugoj osobi tada govorimo o *relaciji* između tih objekata.

### 7.1.1 Tipovi podataka

U tekstu iznad smo pričali o nekim *objektima*, oni u Prologu predstavljaju podatkovne objekte, te je pravilan naziv za njih ***term****.*

Tako *term* može biti konstanta ili varijabla ili složeni term.

**Konstanta**

* Cjelobrojne vrijednosti: 1, 0, 27000, -12, …
* Decimalne vrijednosti 0.101, 3.14, -2.71, …
* Atom: c, mirko, 'Tekst', ***[ ] (lista)*** odnosno sve bilo koji redoslijed alfanumeričkih (uključujući '\_') znakova pri ćemo prvo slovo mora biti malo.

**Varijable**

Varijable su bilo koji niz alfanumeričkih znakova *(čitaj brojeva i slova)* koji započinju velikim slovom ili \_. U slučaju da varijabla započinje sa **\_** tada nju nazivamo anonimnom tj. nebitnom varijablom. *(eng. don't care variable)*

Neki primjeri : **X, Y, Rezultat, \_Rj, \_234, \_** i slično.

Ukoliko se neka varijabla koristi samo jednom u nekoj **rečenici** *(što je u Prolog jeziku naziv za neku funkciju),* odnosno ukoliko ona nije bitna za izvođenje programa ili upita tada se ona zapisuje u obliku \_ znaka.

Varijable u Prologu radi po istom principu kao i u ostalim jezicima, dajemo nekim imenovanim objektima vrijednost, te ih dalje koristimo pod tim imenom.

**Složeni termovi**

*„Strukturirani podatkovni objekti su složeni termovi.“*

Svaki složeni *term* sastoji se od funkcijskog simbola i sekvence od jednog ili više *term*-ova koji se nazivaju argumentima. Funkcijski simbol okarakteriziran je svojim imenom (koje je atom) i svojom arnošću (kratnošću, mjesnošću).

**Što to znači ?**

Primjerice imamo složeni *term* koji ima funkcijski simbol **roditelj** arnosti **2**, s argumentima **X, Y** bi se zapisao kao:

**roditelj( X , Y ).**

Ovime bi se ostvarilo nešto poput funkcije s argumentima, te radi na principu **roditelj(majka, dijete)** ili **roditelj(otac, dijete)**, ili obrnut redoslijed argumenata, ali ukoliko počnemo na jedan način, njega se dalje moramo pridržavati.

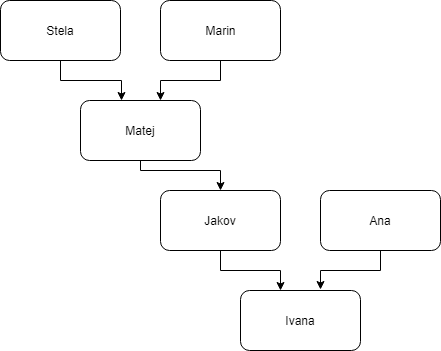
### 7.1.2 Činjenice i pravila

Dva koncepta koja trebamo još poznavati su *činjenice* i *pravila*. Činjenice opisuju elementarne relacije, a pravila određuju nove relacije na temelju već postojećih.

Primjer činjenice:

**roditelj( stela, matej ).**

U primjeru iznad, **roditelj** nam je *relacija*, dok su nam **stela** i **matej** *term*-ovi nad kojima *relacija* **roditelj** djeluje. Time dobivamo činjenicu da je **stela** roditelj od **matej**. Napisani primjer možemo dodatno proširiti te dobijemo obiteljsko stablo. Primjer proširene relacije roditelj:

****

**roditelj( stela, matej ).**

**roditelj( marin, matej ).**

**roditelj( matej, jakov ).**

**roditelj( jakov, ivana ).**

**roditelj( ana, ivana ).**

Slika 46. Obiteljsko stablo

Ovako napisane činjenice u Prologu nazivamo još bazom znanja, budući da sadrže neke konkretne informacije.

Ako želimo ispitati neku relaciju, odnosno dali je neka osoba nekome roditelj, onda iza znakova **?-** pišemo naš upit.

Npr.

**?- roditelj( stela, matej ).**

**yes**

Upit vraća vrijednost **yes** ukoliko zbilja naš upit postoji kao činjenica, odnosno u nekim Prolog jezicima to će biti **true.**

U slučaju da dobijemo **no** *(false)*onda znamo da neka osoba nije roditelj drugoj osobi koju smo naveli u upitu, odnosno postavili smo upit za koji ne postoji činjenica.

Ako želimo saznati tko je roditelj od **matej** onda stavimo na mjesto roditelja, veliko slovo ili neku riječ koja počinje velikim slovom, tj postavimo *varijablu*.

Na taj način, reći ćemo Prologu da nam iz postojećih činjenica, dohvati sve roditelje koje **matej** možda ima.

Na primjer:

**?- roditelj( X, matej).**

**X = stela**

**X= marin**

Iz primjera možemo vidjeti da su **stela** i **marin** roditelji od **matej**. Iz navedenom primjera se dobro vidi moć zaključivanja Prologa, uz uvjet da su sve napisane činjenice ispravne.

Daljnji primjeri:

**?- roditelj(stela, X), roditelj(X,Y).** % Tko je Stelin unuk ?

**?- roditelj(X, jakov), roditelj(X, matej).** % Da li imaju istog roditelja ?

Na ovaj način smo zadali složeni upit Prologu, odvojen zarezom, te na taj način obje tvrdnje moraju biti zadovoljene kako bi se vratio neki rezultat.

### 7.1.3 Pravila

Pomoću pravila se u Prologu definiraju nove, proširene relacije, uz pretpostavku da imamo već prethodno zadane relacije. Tako možemo naš primjer *roditelj*-a proširiti ovako:

**…**

**musko( marin ).**

**musko( matej ).**

**musko( jakov ).**

**zensko( stela ).**

**zensko( ivana ).**

**zensko( ana ).**

**majka(X,Y):-**

**roditelj(X,Y), zensko(X).**

**otac(X,Y):-**

**roditelj(X,Y), musko(X).**

U slučaju iznad, naše nove dvije relacije su **majka** i **otac.** Tako npr. relaciju **majka** možemo ovako interpretirati.

*X je majka Y ako je X roditelj Y i ako je X žensko*

Pravilo za **djeda** bi ovako izgledalo:

*X je djed Z-u ako je X musko i ako je X roditelj Y i ako je Y roditelj Z*

tj.

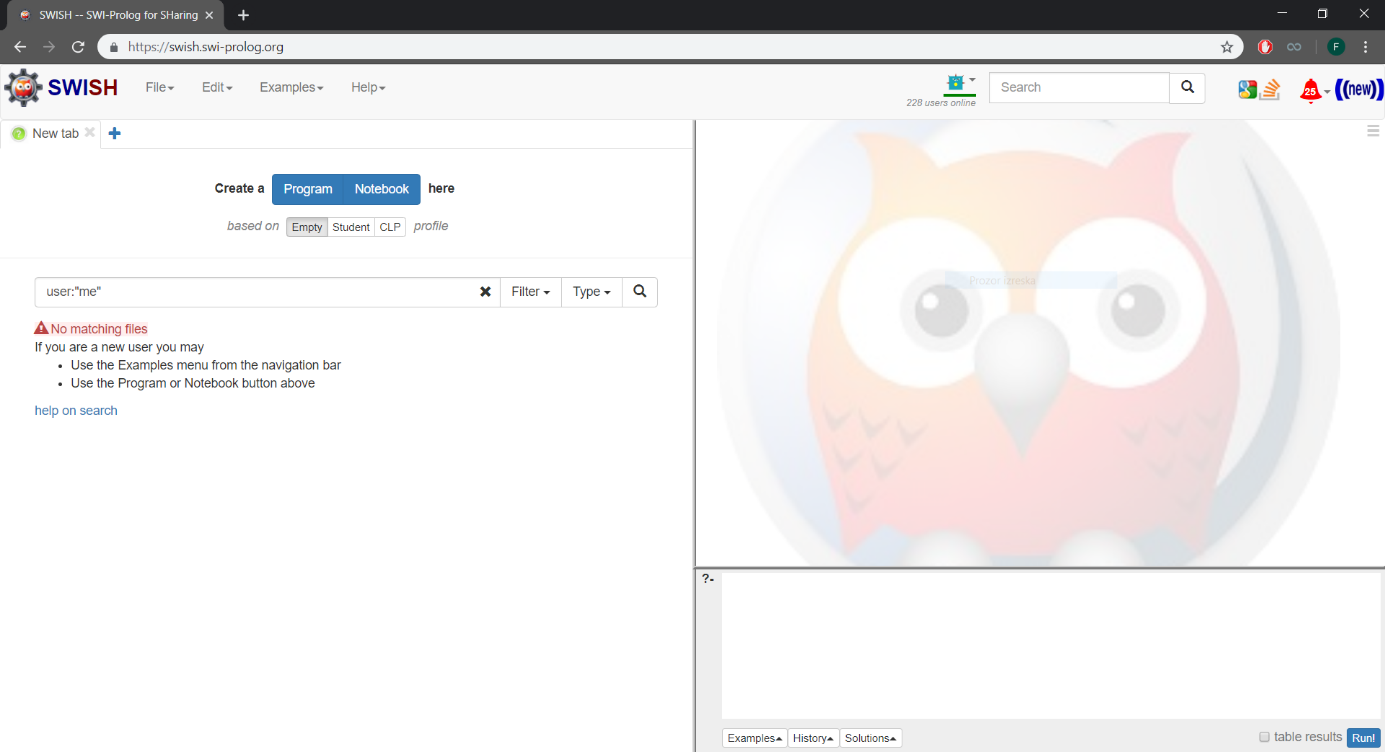
**djed( X, Z ):-**

**musko( X ),**

**roditelj( X, Y ),**

**roditelj( Y, Z ).**

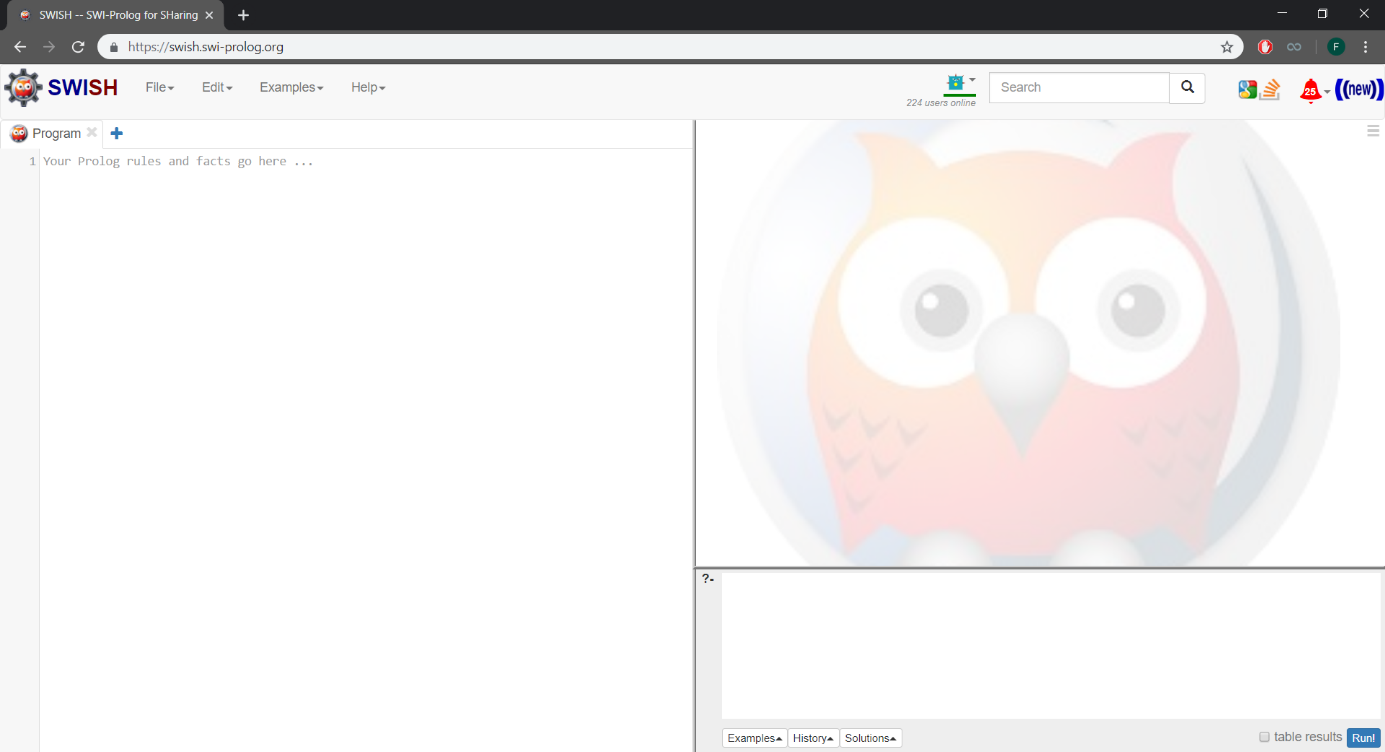
## 7.2 Pisanje i izvršavanje Prolog programa

 Za pisanje Prolog programa možete koristiti besplatni online alat, SWISH dostupan na <https://swish.swi-prolog.org/> lokaciji.

OVDJE PRITISNEMO KAKO BI KRENULI PISATI PROGRAM

Slika 47. SWISH Prolog

Na stranici odaberemo opciju **Program** te nam se otvori sljedeći zaslon, gdje ćemo pisati naš Prolog program.

Kako bi napisani program mogli negdje drugdje koristiti, potrebno ga je preuzeti, a to obavimo na način da pritisnemo na *File* i onda na *Download*. Time smo preuzeli naš napisani Prolog program, i on bi trebao imati ekstenziju .pl.

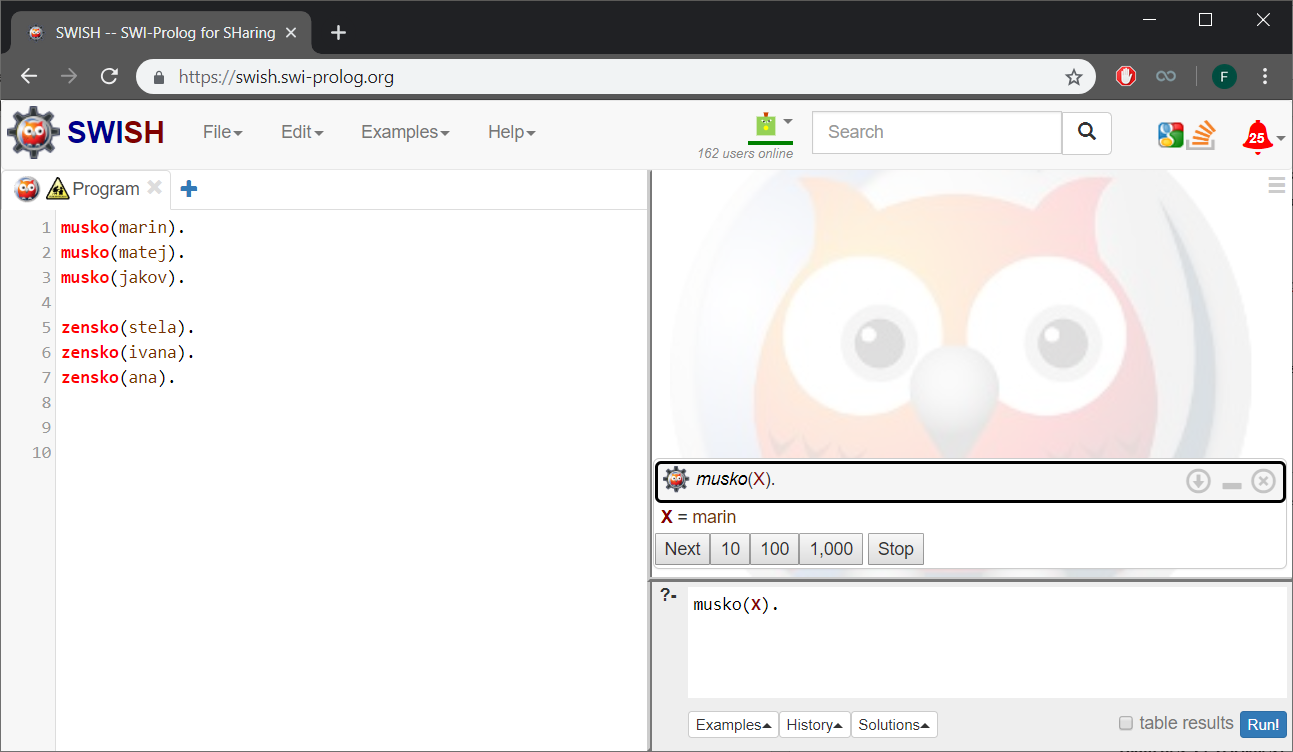
Slika 48. SWISH Prolog – prozor pisanja programa

OVDJE PIŠEMO UPITE POMOĆU KOJEG SE IZVRŠAVA PROLOG PROGRAM

OVDJE SE POKAŽE RJEŠENJE POSTAVLJENOG UPITA

PRITISKOM NA PLAVI GUMB RUN POKREĆEMO NAŠ UPIT

OVDJE PIŠEMO NAŠ PROLOG PROGRAM



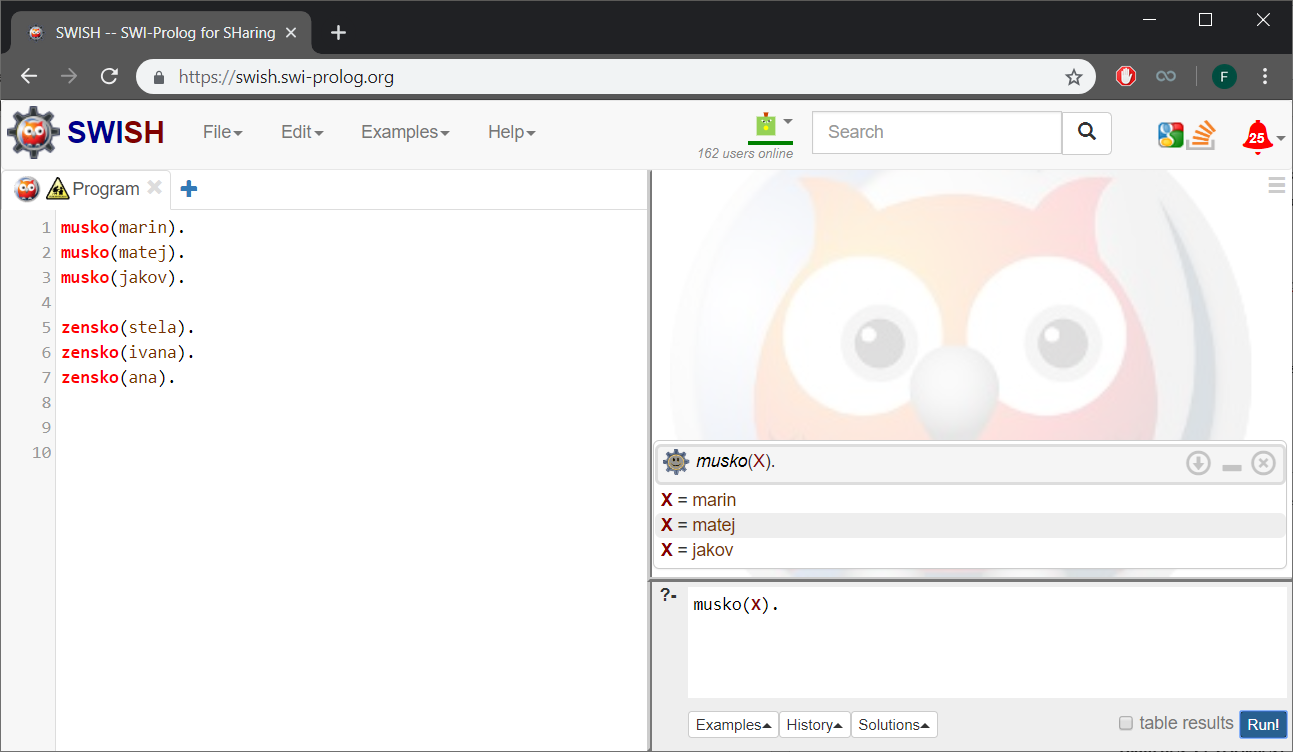
PRIKAŽI JOŠ REZULTATA

KLIKOM NA RUN, DOBIVAMO SLJEDEĆI REZULTAT

OVDJE SMO ZADALI UPIT

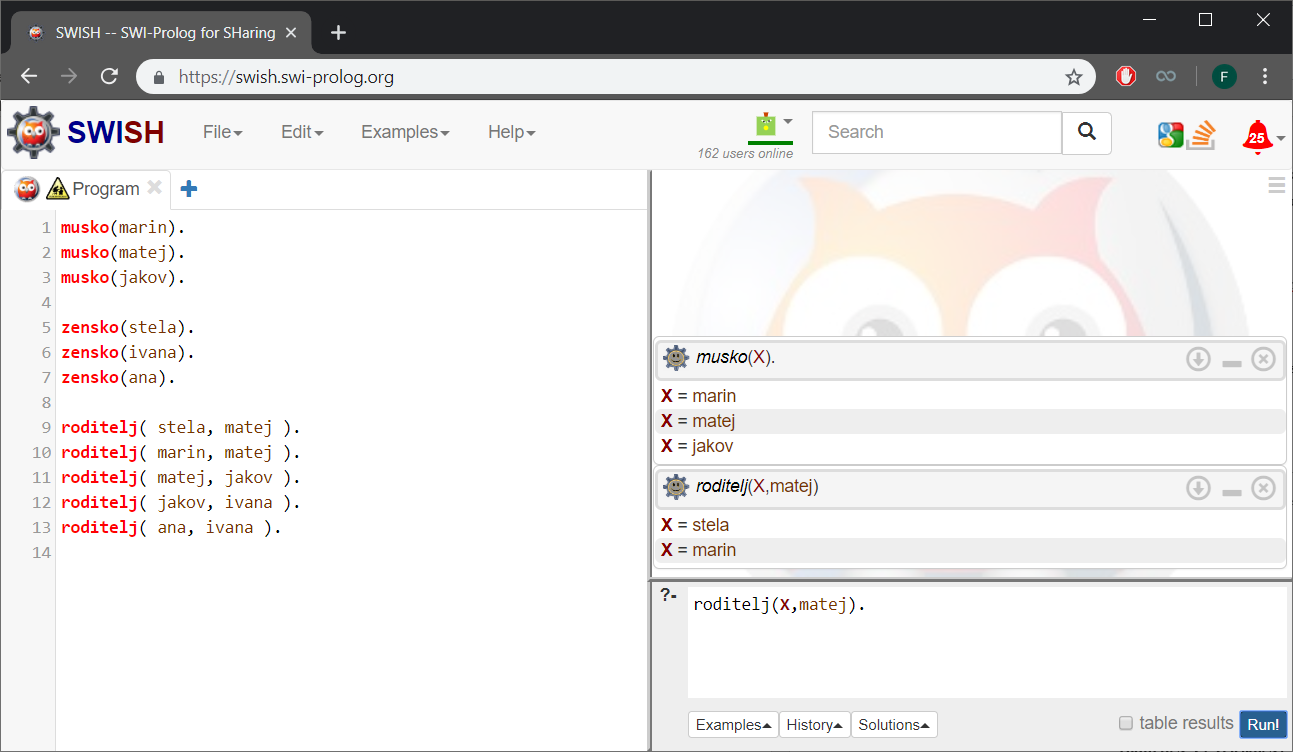
OVDJE SMO NAPISALI NEKE ČINJENICE

Slika 49. SWISH Prolog – pronalazak svih muških 1/2

Izvršavanjem upita, dobivamo nazad sve rezultate koji zadovoljavaju naš upit. Ukoliko ih ima više, neki kompajleri će odmah prikazat sva rješenja, dok ovdje će nam biti prikazano prvo te omogućeno da pomoću gumba ***Next*** dalje idemo kroz rezultate, odnosno prikazujemo sljedećih ***10, 100 ili 1000*** sa pripadajućim gumbima.

POTPUNI REZULTAT IZGLEDA OVAKO

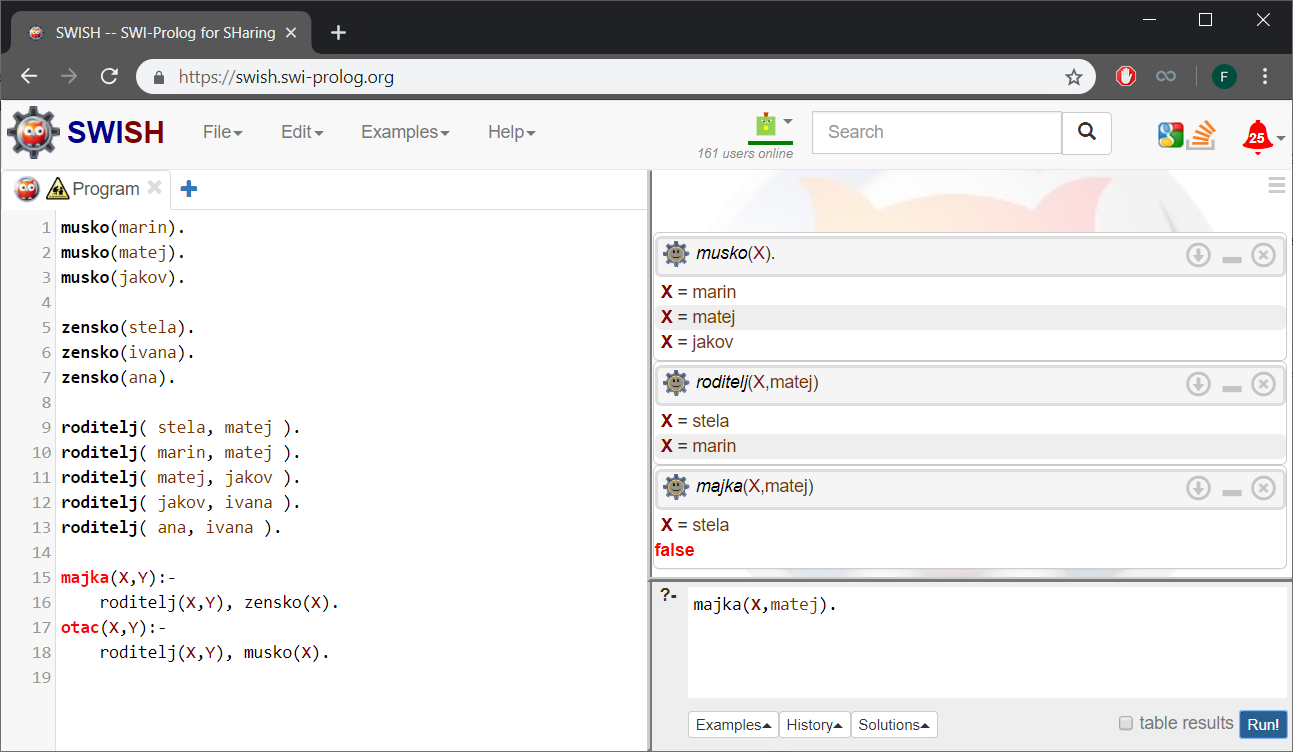
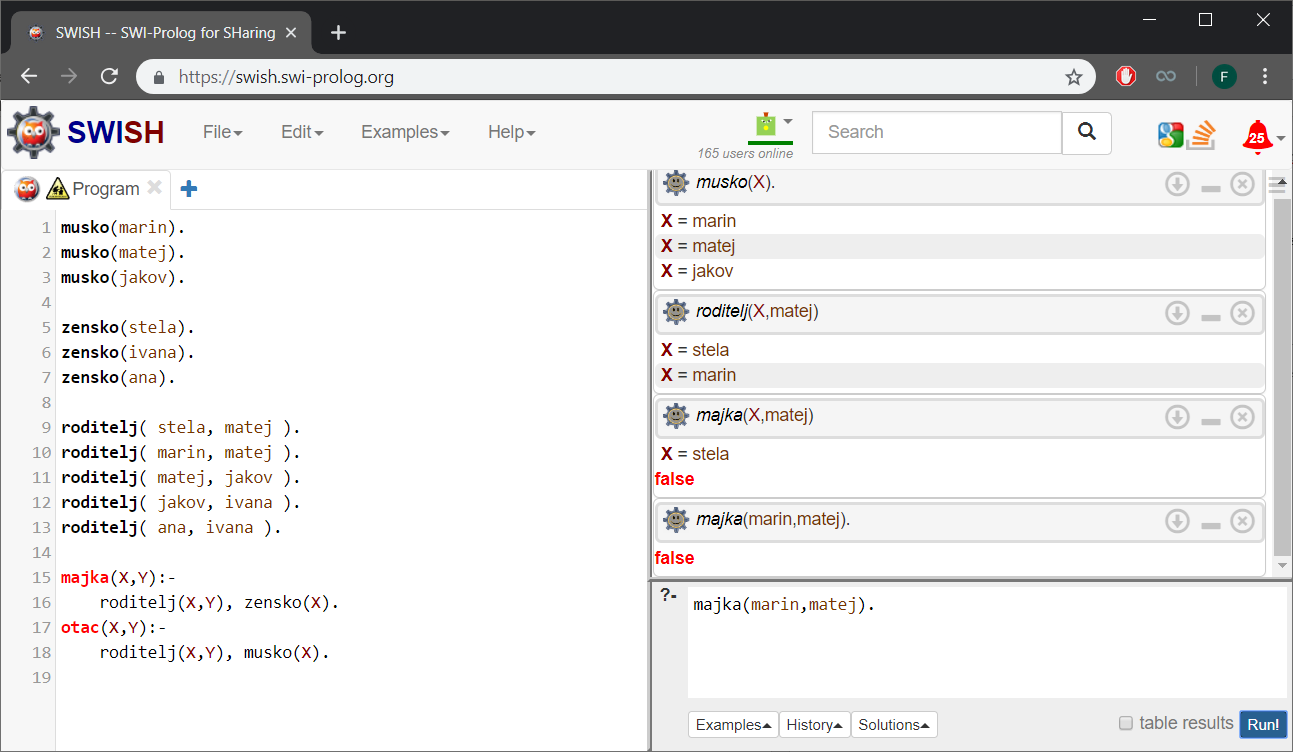
Slika 50. SWISH Prolog – pronalazak svih muških 2/2



TRAŽIMO SVE RODITELJE OD MATEJA

DODALI SMO JOŠ NEKE ČINJENICE

Slika 51. SWISH Prolog – pronalazak Matejevih roditelja



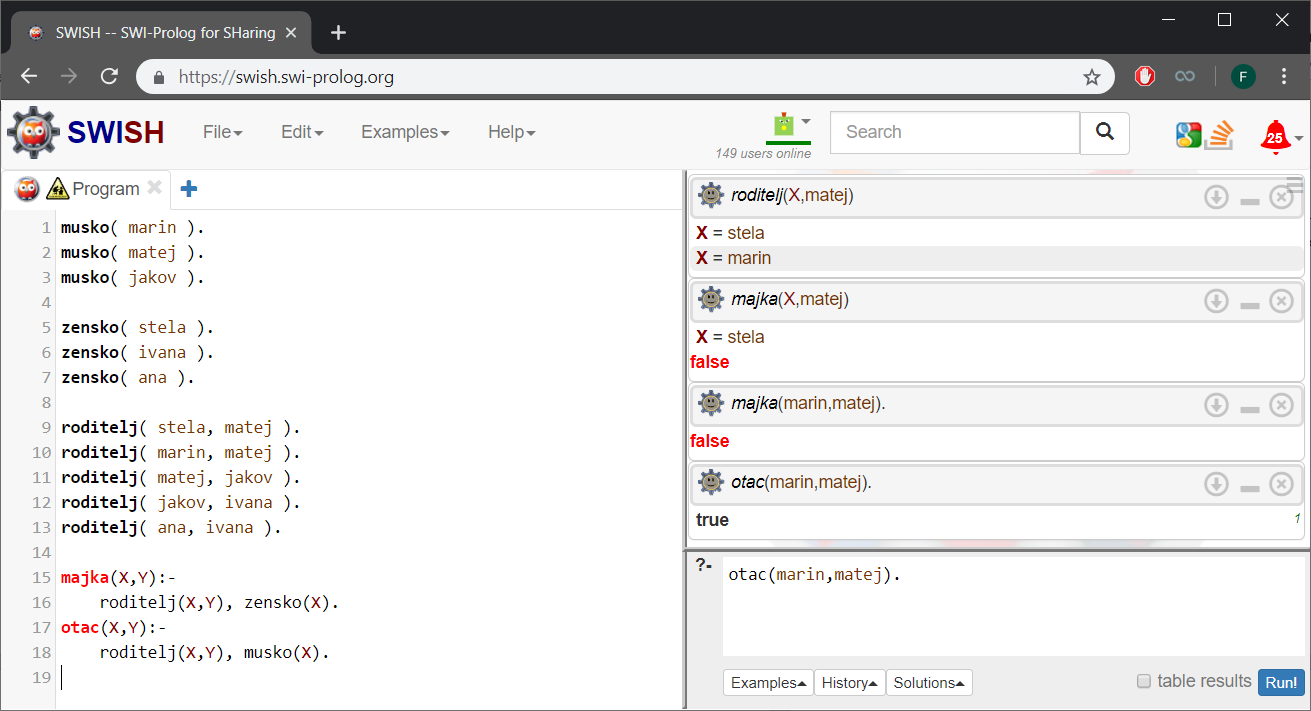
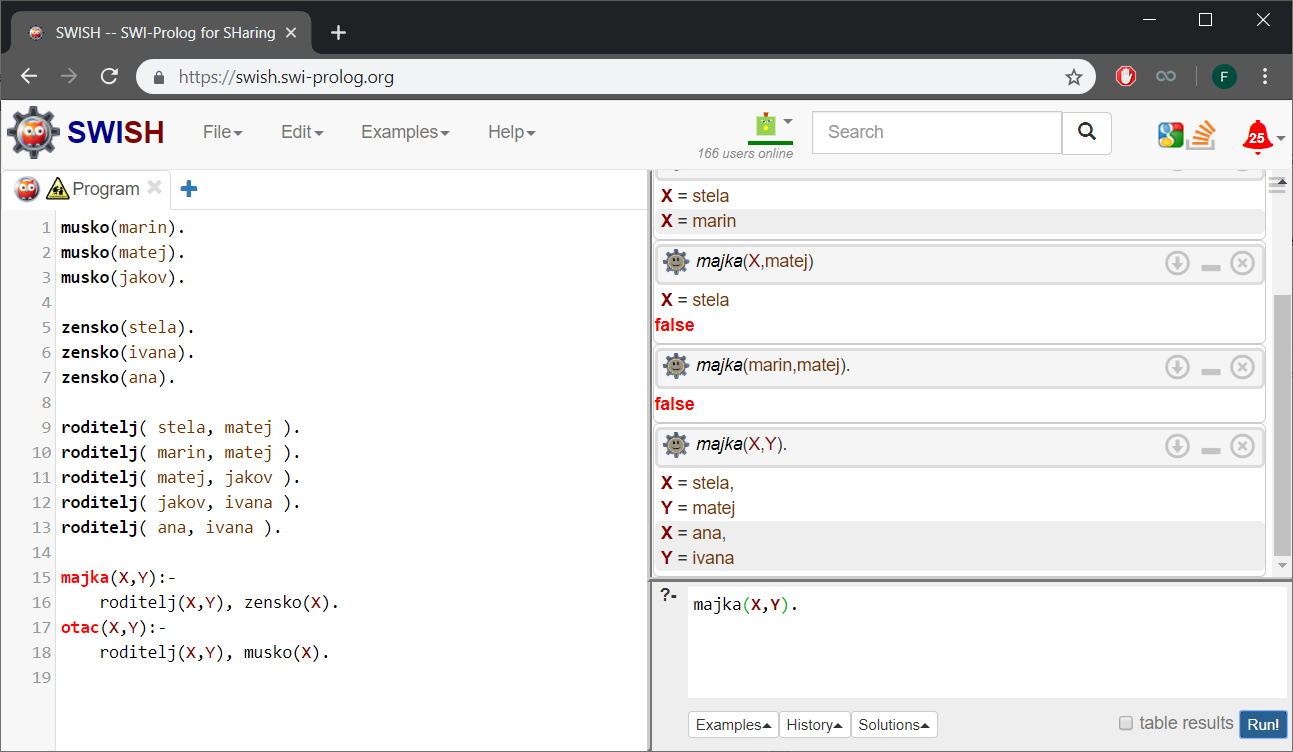
Slika 53. SWISH Prolog – provjera da li je Marin majka od Mateja

DODALI SMO PREDIKATE KOJI PRONALAZE MAJKU I OCA

TRAŽIMO MAJKU OD MATEJA

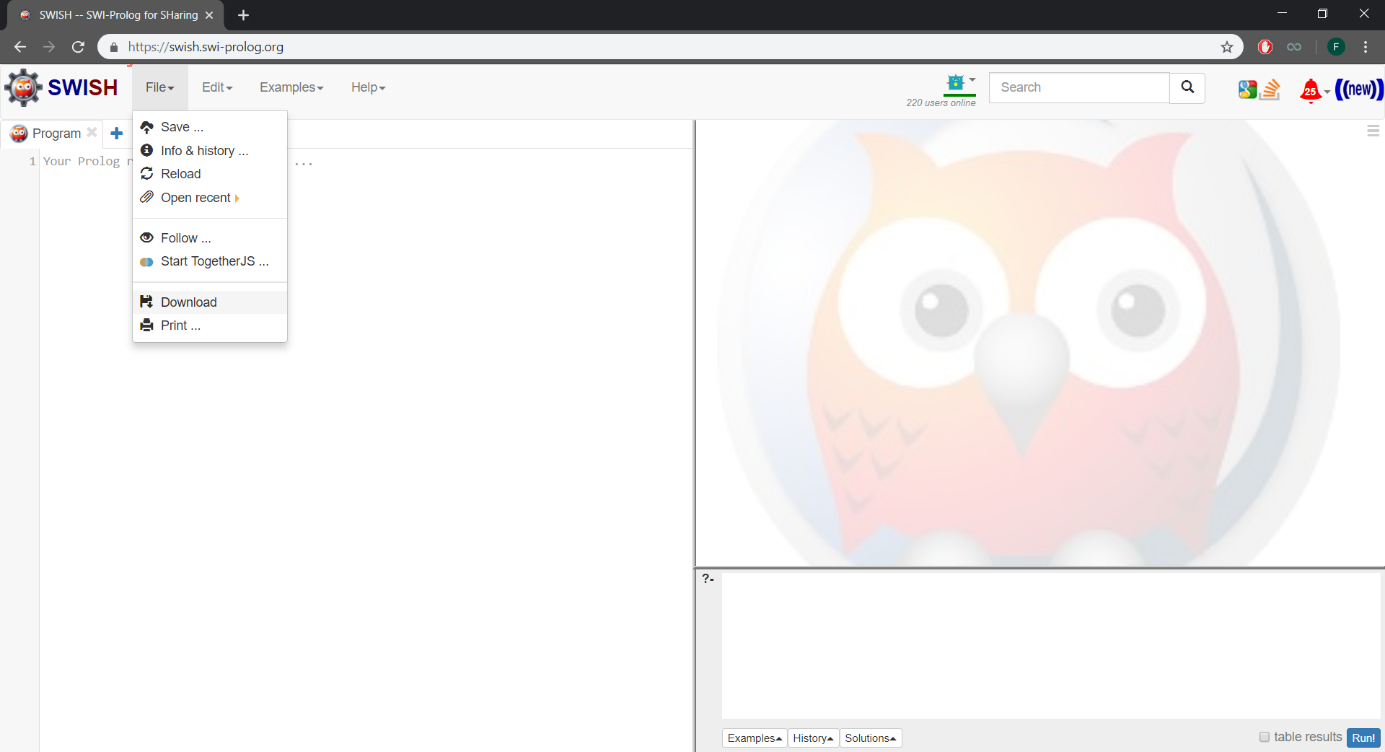
Slika 52. SWISH Prolog – pronalazak Matejeve majke

Upit ***majka(marin, matej)****.* vraća nazad ***false,*** što znači da za taj upit nije pronađeno niti jedno rješenje iz naših činjenica koje ga zadovoljava. U ovom konkretnom slučaju, kako bi netko bio majka od Mateja, ta osoba mu mora biti **roditelj** te još mora biti i **žensko. Marin** je roditelj od Mateja, ali je muško.



Slika 55. SWISH Prolog – pronalaženje svih parova majki i djeteta

Slika 54. SWISH Prolog – provjera da li je Marin otac od Mateja



Slika 56. Spremanje Prolog programa

## 

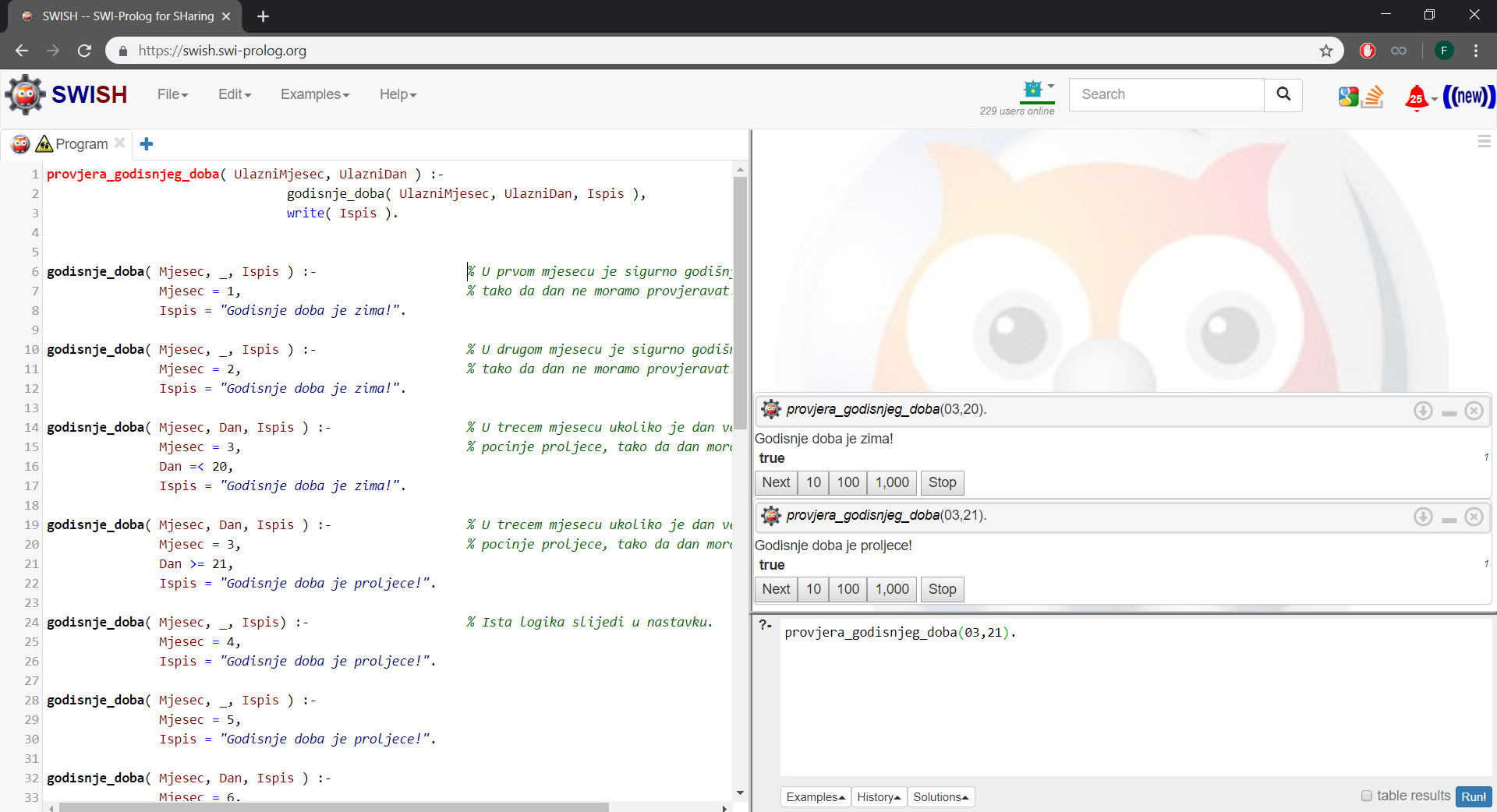
## 7.3 Provjera godišnjeg doba

Prije nego što pokažemo kako se određuje jeli neka temperatura prosječna za to doba godine, pokazati ćemo kako se određuje godišnje doba. Prilikom spremanja temperature s robota u bazu, također smo spremili datum i vrijeme kad smo to bavili. Na temelju datuma ćemo odrediti o kojem se doba godine radi (proljeće, ljeto, jesen ili zima).

Ispod priloženi Prolog kod obavlja taj zadatak, a uz komentare možete zaključiti kako on radi.



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)



PROVJERA GODIŠNJEG DOBA ZA 3 MJESEC I ZA 20 I 21 DAN U TOM MJESECU

Slika 57. Primjer korištenja programa provjera\_godisnjeg\_doba.pl

## 7.4 Provjera prosječne temperature

Sad dok ste shvatili princip Prologa, pisanja programa i upita, onda možemo krenuti na zabavni dio, a to je provjera temperature koju smo dohvatili s robota. Za određivanje prosječne temperature koristimo tri podatka, a to su mjesec, dan i izmjerenu temperaturu te je cjelokupni kod za određivanje toga nešto kompleksniji od prethodnog.

U programu imamo tri doba dana (jutro, dan, noc) za koje možemo provjeriti jeli u određenom mjesecu dohvaćena temperatura ispod, iznad prosjeka ili jeli ona prosječna. Svako od tih doba dana za neki mjesec ima svoju pripadajuću činjenicu u kodu, koju smo mi izračunali za podatke o prosječnim temperaturama u Varaždinu za 2017. godinu.

|  |  |  |  |
| --- | --- | --- | --- |
| Mjesec | Noć | Jutro | Dan |
| 8. | 14.7 | 21.4 | 28.0 |
| 9. | 9.8 | 14.2 | 18.5 |
| 10. | 4.8 | 11.7 | 18.5 |
| 11. | 1.2 | 5.6 | 10.0 |
| 12. | -2.6 | 2.4 | 7.3 |

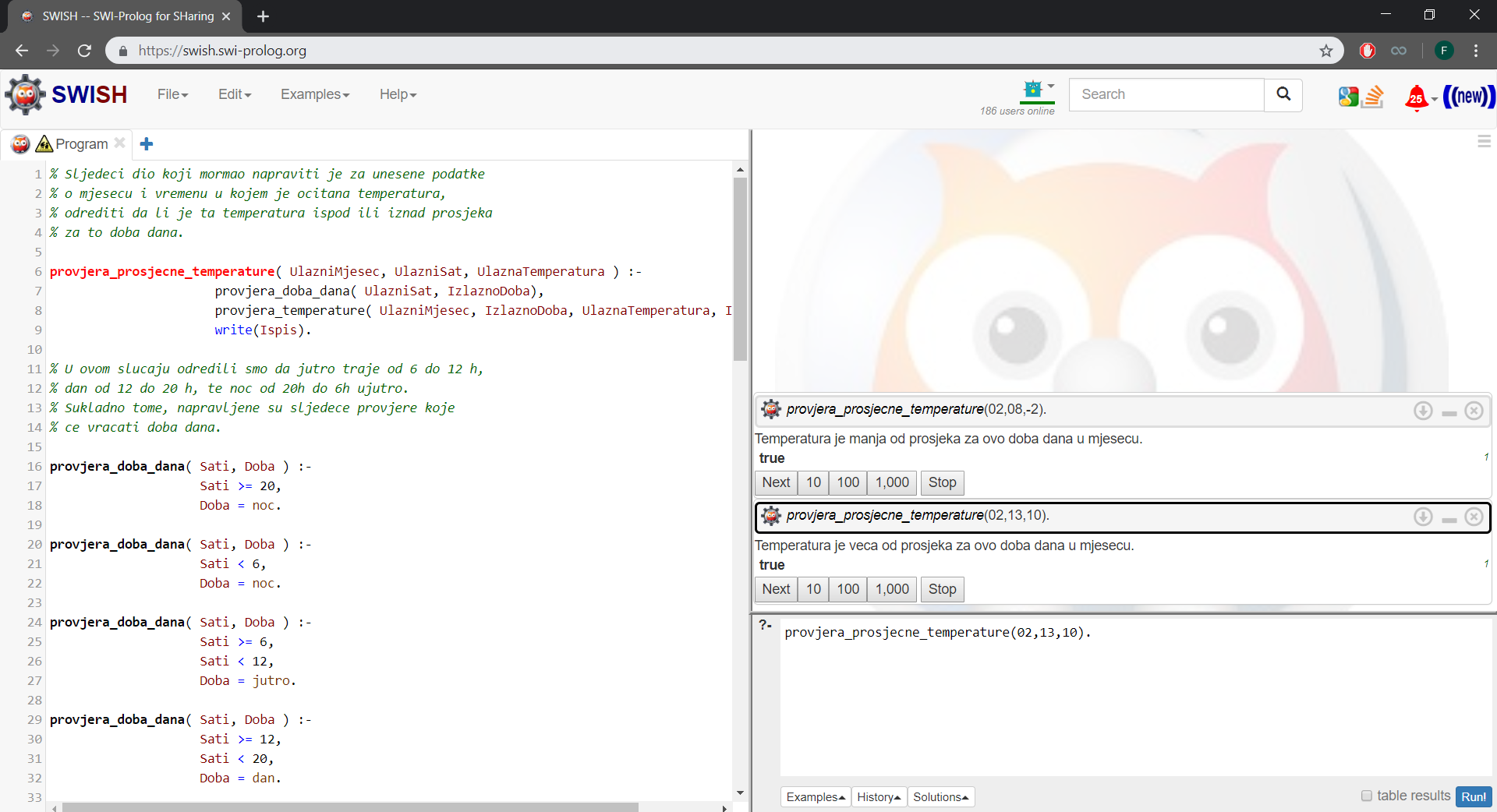
|  |  |  |  |
| --- | --- | --- | --- |
| Mjesec | Noć | Jutro | Dan |
| 1. | -8.5 | -4.8 | -1.1 |
| 2. | -0.4 | 4.0 | 8.4 |
| 3. | 2.6 | 9.3 | 15.9 |
| 4. | 4.6 | 10.4 | 16.2 |
| 5. | 9.5 | 15.8 | 22.0 |
| 6. | 14.4 | 20.7 | 26.9 |
| 7. | 15.1 | 21.8 | 28.4 |

Graf 1. Prosječne temperature po mjesecima

U samom Prolog programu imate točne opise i komentare koja naredba što obavlja.



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)



PROVJERA TEMPERATURE ZA 2 MJESEC U 8 I U 13 SATI. TEMPERATURE SU -2 I 10

Slika 58. Primjer korištenja programa provjera\_prosjecne\_temperature.pl

## 7.5 Pozivanje Prolog programa iz PHP-a

Ovime smo napisali dva Prolog programa koja zaključuju o kojem se godišnjem doba radi te jeli dohvaćena temperatura prosječna ili ne. Sad je jedino potrebno povezati i primijeniti taj napisani Prolog kod. Mi smo se odlučili za PHP koji se koristi za obradu podataka na raznim Internetskim stranicama.

Ovdje vam nećemo objašnjavati kako i na koji način radi PHP jezik već samo kako se poziva naši Prolog programi.

**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

Naredbu pozivanja i izvršavanja Prolog programa dijelimo na tri djela, prvi dio pomoću kojeg definiramo putanju do .exe datoteke, odnosno samog programa koji izvršava naše naredbe (kompajler), drugi dio koji počinje s -f iza kojeg definiramo Prolog datoteku u kojoj se nalazi naš kod, a zadnji, treći dio koji počinje s -g iza kojeg pozivamo neku od naših „metoda“ iz našeg Prolog programa.

U zadnjem djelu, unutar zagrada postavljamo naše vrijednosti ili varijable koje sadrže vrijednosti koje želimo obraditi. Ovdje je bitno da nemamo nikakvih razmaka u zagradi jer se inače neće dobro pozivat Prolog program.

Sad je još jedino potrebno napisanu naredbu pozvati. Postoji nekoliko načina, a najjednostavnija je *shell\_exec()*, u koju postavimo našu napisanu naredbu. Ona pozove kompajler, Prolog program koji smo zadali, te „metodu“ s postavljenim podacima, a vrati ono što *write()* naredba ispiše u programu.

Kako bi na stranici prikazali dohvaćenu poruku, u PHP-u poruku ispisujemo s naredbom *echo*.

## 7.6 Usporedba PHP i Prolog rješenja

Programi koji su napisani u Prolog jeziku također se mogu napisati u PHP programskom jeziku, no tada se postavlja pitanje gdje ima više smisla napisati takav program. U prethodnom dijelu dokumenta smo pokazali kako izgleda kod napisan u Prolog jeziku, a sada ćemo vam priložiti iste te programske odsječke napisane u PHP-u na temelju kojih ćemo ustanoviti gdje ima smisla koristiti Prolog jezik, a gdje ne.

### 7.6.1 Provjera godišnjeg doba u PHP-u



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

Provjera godišnjeg doba u PHP jeziku je vrlo jednostavno izvedeno te ima jednak broj linija kao i Prolog program, osim toga sam kod u PHP- je dovoljno pregledan i jasno strukturiran, a po potrebi se može dodatno smanjiti ali se tada vizualna strukturiranost gubi.

U takvom slučaju Prolog program nema nekakve prednosti, budući da se dobije jedno te isti rezultat. Osim toga Prolog program će se tada sporije izvoditi jer je potrebno otvoriti poseban kompajler i tamo pokretati napisani Prolog program, što nije idealno.

Zaključujemo da bi u takvim okolnostima bilo dobro izbjegavati pisanje Prolog programa te bi se trebao koristiti postojeći programski jezik.

### 7.6.1 Provjera prosječne temperature u PHP-u



**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)

U takvom primjeru, kada imamo veliki broj „ *if-else if* “ provjera, Prolog program znatno smanji broj napisanih linija, tj. s nekakvih 275 linija na tek 95 linija. Osim toga tako napisan program je znatno čitljiviji te je bolje strukturiran, a time i razumljiviji.

Tako velika prednost postignuta je na napisanim činjenicama za temperature po dobu dana u određenom mjesecu, odnosno našoj bazi znanja. U PHP-u je potrebno za takvu provjeru napisati nekoliko „ *if* “ provjera, dok u Prologu napišemo u jednom redu takvu činjenicu te program sam zaključi po tome što treba učiniti. U prijevodu, kada imamo veliki broj provjera, tada se s jednostavnim činjenicama kod može znatno smanjiti te nam na neki način zamijeni „ *if* “naredbu.

To je samo jedan dobar primjer gdje je Prolog koristan, ali ima znatno veći broj takvih primjera, potrebno je samo malo vremena i truda uložiti kako bi se takvo nešto postiglo.

## 7.7 Dodatni Prolog materijali

Dosad napisani sadržaj je dovoljan za izradu osnovnih Prolog programa, ali za nešto više potrebna su dodatna znanja i vještine, a ovime ćemo vam pružati nekakav dodatni sadržaj čime si možete proširiti već postojeće znanje iz Prologa. Osim toga, priloženi sadržav vam možda dodatno pojasni nekakve nedoumice koje ste dosad imali.

### 7.7.1 Liste

Liste kao i u svakom programskom jeziku su jedni od glavnih podatkovno strukturnih elemenata te je bitno da se one savladaju. U Prologu su liste atomi označene s „ [] “ simbolima, a mogu biti i složeni term-ovi s funkcijskim simbolom „ . “. U većini slučajeva se koriste atomi s oznaka „ [] “.

Npr.

[ 1 , 2 , 3 ]

Kod lista je bitno znati da se njen sadržaj dijeli na Glavu (G) i na Rep (R). Glava je uvijek prvi element iz liste, a Rep je cijeli onaj ostatak bez prvog elementa. Na primjeru iznad, Glava je 1, a Rep [ 2 , 3 ]. Iz Repa [ 2 , 3 ] Glava je 2, a Rep [ 3 ]. Kao zadnje, iz liste [3] Glava je 3, a Rep [].

### 7.7.2 Rekurzije

U programskom svijetu **rekurzije** su programi koji sami sebe pozivaju, i to neki određeni broj puta jer mora postojati neki **granični uvjet** gdje će se zaustaviti. Međutim u Prologu nemamo takve postojeće konstrukcije koja to omogućuje.

Kako bi takvo nešto postigli, koristimo rekurzivne relacije, a da bi vam to bilo jasnije koristit ćemo primjer *roditelj.*

% predak je roditelj…

predak( X, Y ) :-

roditelj( X, Y ).

% … ali i roditelj od bilo kojeg drugog pretka

predak( X, Z ) :-

roditelj( X, Y ),

predak( Y, Z ).

Relacija *predak* je opisana sa dva stavka što znači da vrijedi ako su zadovoljeni svi od njih. Više stavaka koji definiraju jednu relaciju čine *proceduru* te je redoslijed stavki u proceduri bitan.

Budući da znamo kako liste funkcioniraju, mogli bi napraviti ispis pojedinih elemenata iz liste, a za takvo nešto potrebna nam je rekurzija. Primjer možete sami isprobati.

elementi( [ G | R ] ):- %cjelokupna lista gdje se izdvaja prvi element od ostatka

writeln( G ), %ispis glave (writeln - idući ispis ide u novi red)

elementi( R ). %ponovno pozivanje predikata *elementi* s listom bez glave tako dugo dok ima nešto za ispisat

Postoje slučajevi u kojima se rekurzija neće niti početi izvršavati te nam je stoga potreban uvjet pomoću kojeg ispisujemo rješenje. U ostalim slučajevima potrebno je izvršavanje rekurzije zaustaviti u točno određenom trenutku gdje je također potreban izlazni uvjet.

Kako bi vam to bilo jasnije ispod smo vam priložili kratke animacije gdje je vizualno prikazano korištenje takvog uvjeta. U animacijama se međusobno zbrajaju brojevi iz liste.

**Granični uvjet** je istoimeni predikat čiji argumenti čine logičku granicu izvođenja. Odnosno ukoliko tijekom izvođenja programa dođe do zadovoljenja tog predikata, to će omogućiti programu izlazak iz rekurzije.

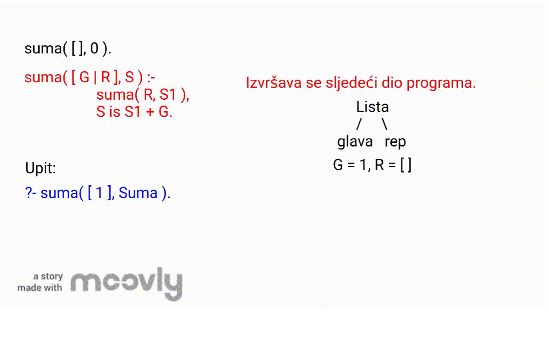
Primjer s praznom listom [ ]:

[Slika na kojoj se prikazuje snimka zaslona

Opis je automatski generiran](https://imgur.com/a/GKVl0Lv)

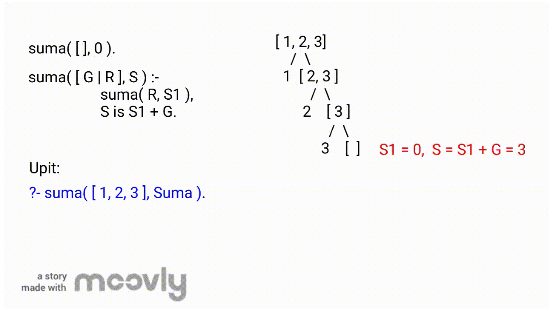
Slika 59. Veza na animaciju s primjerom prazne liste

Primjer s listom [ 1 ]:

[](https://imgur.com/a/v6Qo4I6)

Slika 60. Veza na animaciju s primjerom liste s jednim elementom

Primjer s listom [ 1 , 2 , 3 ]:

[](https://imgur.com/a/QwEjLmX)

Slika 61. Veza na animaciju s primjerom liste s tri elementa

(**Ctrl + klik** na sliku kako biste otvorili animacije u pregledniku)

U prethodnim primjerima imali smo predikat koji je služio zbrajanju elemenata liste. Granični uvjet u svakom od tih primjera je bio **suma( [ ], 0 ).** Taj predikat je značio da ukoliko tijekom izvođenja programa, dođemo do toga da imamo praznu listu, suma te liste će biti **0 (nula).**

****

Primjerice ako imamo listu sa jednim elementom, **[1]**, granični uvjet nije zadovoljen te ulazimo u sljedeći predikat, te dijelimo tu listu na glavu od jednom elementa – **1** i preostalog repa, koji će biti **prazna lista [ ].**

Daljnjim izvođenjem koda izvršava se ponovno taj predikat gdje je lista zapravo preostali rep, koji je u ovom slučaju prazna lista, te se time zadovoljava **granični uvjet** i program zna da je kraj pozivanja rekurzija, te se vraća istim putem nazad. Time će **S1 postati 0,** te na samom kraju **Suma** će biti **G ( 1 ) + S1 ( 0 ) = 1.**

Ovo gradivo treba malo vježbe i strpljenja da sjedne, te ukoliko Vam još uvijek nije jasno, najbolje što možete je pregledati ponovno drugu animaciju (Slika 53.). Istom logikom kao i kod Slika 54. se program izvršava za bilo koji broj elemenata liste.

### 7.7.3 Posebni ugrađeni predikati

**(**Potrebno je kliknuti dva puta na okvir koda te će se cjelokupni kod prikazati za kopiranje.)