1 početak

Da se bućnemo

Kotlin podiže talase.

Od svog prvog izdanja, Kotlin je zadivio programere svojom ***pristupačnom sintaksom, sažetošću, fleksibilnošću i snagom***. U ovoj knjizi, naučićemo vas kako da **gradite aplikacije u Kotlinu**, a počećemo tako što ćete napraviti jednostavnu aplikaciju i pokrenuti je. Usput ćete upoznati deo osnovne Kotlinove sintakse, kao što su *naredbe, petlje* i *uslovno grananje*. Vaše putovanje je upravo počelo...

Dobrodošli u Kotlingrad

Kotlin je osvojio programerski svet na prečac. Uprkos tome što je jedan od najmlađih programskih jezika u kraju, mnogi programeri su ga izabrali za svoj jezik. Šta to čini Kotlin toliko posebnim?

Kotlin ima mnoge osobine modernog jezika koje ga čine privlačnim za programere. O njima ćete saznati više kasnije u knjizi, a zasad, evo nekih najupečatljivijih.

Jasan je, koncizan i čitak

Za razliku od nekih drugih jezika, Kotlinov kôd je veoma sažet i samo jednim redom možete izvesti moćne zadatke. On nudi prečice za uobičajene akcije tako da ne morate da pišete mnogo uobičajenog koda koji se ponavlja, i ima bogatu biblioteku funkcija koje možete da koristite. A pošto ima manje koda kroz koji morate da se probijate, brže se čita, piše i razume, ostavljajući vam više vremena za druge stvari.

Možete koristiti i objektno orijentisano i funkcionalno programiranje

Ne možete da odlučite da li da učite objektno orijentisano ili funkcionalno programiranje? Zašto ne biste oba? Kotlin vam omogućava da pišete objektno orijentisan kôd koji koristi klase, nalseđivanje i polimorfizam, kao u Javi. Ali, on podržava i funkcionalno programiranje, pružajući vam najbolje iz oba sveta.

Bezbedni ste sa kompajlerom

Niko ne voli nebezbedan kôd pun grešaka, a Kotlinov kompajler se izuzetno trudi da kôd koji pišete bude što je moguće čistiji, sprečavajući mnoge greške koje se mogu javiti u drugim programskim jezicima. Kotlin, na primer, ima statične tipove podataka, pa ne možete obavljati neodgovarajuće akcije na pogrešnom tipu promenljive i srušiti kôd. Uglavnom i ne morate sami izričito da navodite tip jer kompajler može da ga utvrdi umesto vas.

Kotlin bukvalno eleminiše greške kakve se redovono javljaju u drugim programskim jezicima. To znači da imate bezbedniji, pouzdaniji kôd i trošite manje vremena na hvatanje bubica.

Znači, Kotlin je moderan, moćan i prilagodljiv programski jezik koji nudi mnoge prednosti. Ali tu nije kraj.

Kotlin možete koristiti gotovo svuda

Kotlin je toliko moćan i fleksibilan da ga možete koristiti kao jezik opšte namene u raznim kontekstima, zahvaljujući činjenici da možete da ***birate za koju ćete platformu kompajlirati Kotlinov kôd***.

Javine virtuelne mašine (JVM)

Kotlinov kôd se može kompajlirati u JVM (Javina virtuelna mašina, engl. *Java Virtual Machine*) binarni kôd, pa Kotlin možete da koristite praktično svuda gde možete koristiti Javu. Kotlin je stoprocentno interoperabilan sa Javom, pa postojeće Javine biblioteke možete koristiti sa njim. Ako radite na aplikaciji koja sadrži mnogo Javinog koda, ne morate da bacite sav taj stari kôd; novi Kotlinov kôd će raditi uz njega. A ako hoćete da koristite Kotlinov kôd koji ste napisali unutar Jave lako možete i to.

Mogućnost da birate za koju ćete platformu kompajlirati kôd znači da Kotlinov kôd može da se pokrene na serverima, u oblaku, u čitačima veba, na mobilnim uređajima itd.

Android

Pored drugih jezika, kao što je Java, Kotlin ima prvoklasnu podršku za Android. Kotlin je potpuno podržan u Android studiju i možete maksimalno iskoristiti njegove prednosti kada programirate aplikacije za Android.

JavaScript na klijentskoj i serverskoj strani

Kotlinov kôd možete prevoditi – ili kompajlirati – i u JavaScript, pa ga možete pokrenuti u čitaču veba. Možete ga koristiti da biste radili i sa klijentskom i sa serverskom tehnologijom, kao što je WebGL ili Node.js.

Matične aplikacije

Ako želite da pišete kôd koji će se brzo izvoditi na manje moćnim uređajima, Kotlinov kôd možete kompajlirati direktno u matični mašinski kôd. To omogućava da pišete kôd koji će raditi, na primer, u iOS-u ili Linuxu.

U ovoj knjizi ćemo se fokusirati na pravljenje Kotlinovih apliakcija za JVM, pošto je to najjednostavniji način da shvatite jezik. Kasnije ćete moći da primenite stečeno znanje na drugim platformama.

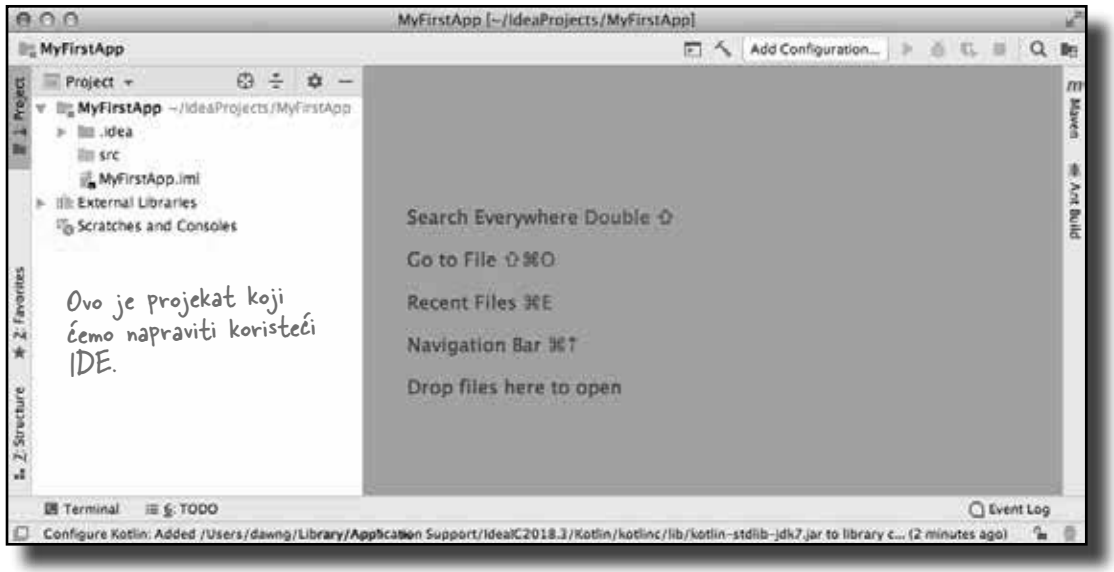
Uronimo!

Iako pravimo aplikacije za Javine virtuelne mašine, ne morate vladati Javom da biste izvukli maksimum iz ove knjige. Pretpostavljamo da imate uopšteno programersko iskustvo i to je sve.

Šta ćemo raditi u ovom poglavlju

U ovom poglavlju vam pokazujemo kako da napravite osnovnu aplikaciju u Kotlinu. Da bismo to uradili, proći ćemo kroz više koraka:

1. **Pravljenje novog Kotlinovog projekta.**
   Počećemo tako što ćemo instalirati IntelliJ IDEA (Community Edition), besplatno integrisano razvojno okruženje (engl. *integrated development environment*, IDE) koje podržava razvoj Kotlinovih aplikacija. Potom ćemo iskoristiti IDE da bismo izgradili nov Kotlinov projekat:



2. **Dodavanje funkcije koja prikazuje tekst.**
   Dodaćemo projektu novu Kotlinovu datoteku, a zatim napisati jednostavnu funkciju `main` koja će prikazivati tekst „Pow!“

3. **Ažuriranje funkcije tako da obavlja više stvari.**
   Kotlin sadrži osnovne strukture jezika kao što su naredbe, petlje i uslovno grananje. Njih ćemo upotrebiti da bismo izmenili funkciju tako da radi više stvari.

4. **Isprobavanje koda u Kotlinovom interaktivnom komandnom okruženju.**
   Najzad, videćemo kako se testiraju odlomci koda u Kotlinovom interaktivnom komandnom okruženju (REPL-u).

IDE ćemo instalirati nakon što isprobate narednu vežbu.

Naoštrite olovku

Znamo da vas još uvek nismo naučili nikakvom Kotlinovom kodu, ali probajte da pogodite šta svaki red narednog koda radi. Za početak, mi smo rešili prvi.

```
val name = "Misty"  Deklariše promenljivu 'name' i daje joj vrednost „Misty".
val height = 9

println("Hello")
println("My cat is called $name")
println("My cat is $height inches tall")

val a = 6
val b = 7
val c = a + b + 10
val str = c.toString()

val numList = arrayOf(1, 2, 3)
var x = 0
while (x < 3) {
    println("Item $x is ${numList[x]}")
    x = x + 1
}

val myCat = Cat(name, height)
val y = height - 3
if (y < 5) myCat.miaow(4)

while (y < 8) {
    myCat.play()
    y = y + 1
}
```

Naoštrite olovku Rešenje

Znamo da vas još uvek nismo naučili nikakvom Kotlinovom kodu, ali probajte da pogodite šta svaki red narednog koda radi. Za početak, mi smo rešili prvi.

```
val name = "Misty"          Deklariše promenljivu 'name' i daje joj vrednost „Misty".
val height = 9              Deklariše promenljivu 'height' i daje joj vrednost 9.

println("Hello")            Ispisuje „Hello" na standardnom izlazu.
println("My cat is called $name")   Ispisuje „My cat is called Misty".
println("My cat is $height inches tall")   Ispisuje „My cat is 9 inches tall".

val a = 6                   Deklariše promenljivu 'a' i daje joj vrednost 6.
val b = 7                   Deklariše promenljivu 'b' i daje joj vrednost 7.
val c = a + b + 10          Deklariše promenljivu 'c' i daje joj vrednost 23.
val str = c.toString()      Deklariše promenljivu 'str' i daje joj tekstualnu vrednost „23".

val numList = arrayOf(1, 2, 3)   Pravi niz koji sadrži vrednosti 1, 2 i 3.
var x = 0                   Deklariše promenljivu 'x' i daje joj vrednost 0.
while (x < 3) {             Prolazi kroz petlju sve dok je x manje od 3.
    println("Item $x is ${numList[x]}")   Ispisuje indeks i vrednost svake stavke u nizu.
    x = x + 1               Vrednosti x dodaje 1.
}                           Ovo je kraj petlje.

val myCat = Cat(name, height)   Deklariše promenljivu 'myCat' i pravi objekat Cat.
val y = height - 3          Deklariše promenljivu 'y' i daje joj vrednost 6.
if (y < 5) myCat.miaow(4)   Ako je y manje od 5, mačka (Cat) treba da mjaukne
                            (miaow) 4 puta.
while (y < 8) {             Prolazi kroz petlju sve dok je y manje od 8.
    myCat.play()            Tera mačku (Cat) da se igra (play).
    y = y + 1               Vrednosti Y dodaje 1.
}                           Ovo je kraj petlje.
```

Vi ste ovde.

- [ ] **Pravljenje aplikacije**
- [ ] **Dodavanje funkcije**
- [ ] **Ažuriranje funkcije**
- [ ] **Korišćenje REPL-a**

Instalirajte IntelliJ IDEA (Community Edition)

Najjednostavniji način za pisanje i pokretanje Kotlinovog koda jeste korišćenje IntelliJ IDEA (Community Edition). To je besplatno integrisano razvojno okruženje koje su napravili JetBrains, ljudi koji su smislili Kotlin, a dobijate ga sa svim što vam je potrebno za razvoj Kotlinovih aplikacija, uključujući:

Oblast za unošenje koda
Oblast za unošenje koda sadrži opciju dovršavanja koja vam pomaže u pisanju koda, i omogućava formatiranje i isticanje pomoću boja da bi kôd bio čitljiviji. Daje vam i predloge za poboljšanje koda.

Ugrađene alate
Možete kompajlirati i pokretati kôd koristeći brze i jednostavne prečice.

Kotlinov REPL
Imate lak pristup Kotlinovom REPL-u, koji dozvoljava da isprobate odlomke koda izvan glavnog koda.

Kontrola verzije
IntelliJ IDEA okruženja sa glavnim sistemima za kontrolu verzije kao što su Git, SVN, CVS itd

Tu su i mnoge druge mogućnosti koje vam olakšavaju programerski život.

Da biste pratili zadatke iz ove knjige, treba da instalirate IntelliJ IDEA (Community Edition). IDE možete preuzeti odavde:

https://www.jetbrains.com/idea/download/index.html

Obavezno odaberite opciju za preuzimanje besplatnog izdanja IntelliJ IDEA, Community Edition.

Kada instalirate IDE, otvorite ga. Trebalo bi da vidite uvodni ekran IntelliJ IDEA. Spremni ste za pravljenje svoje prve aplikacije u Kotlinu.

Ovo je uvodni ekran IntelliJ IDEA.

- [ ] **Pravljenje aplikacije**
- [ ] **Dodavanje funkcije**
- [ ] **Ažuriranje funkcije**
- [ ] **Korišćenje REPL-a**

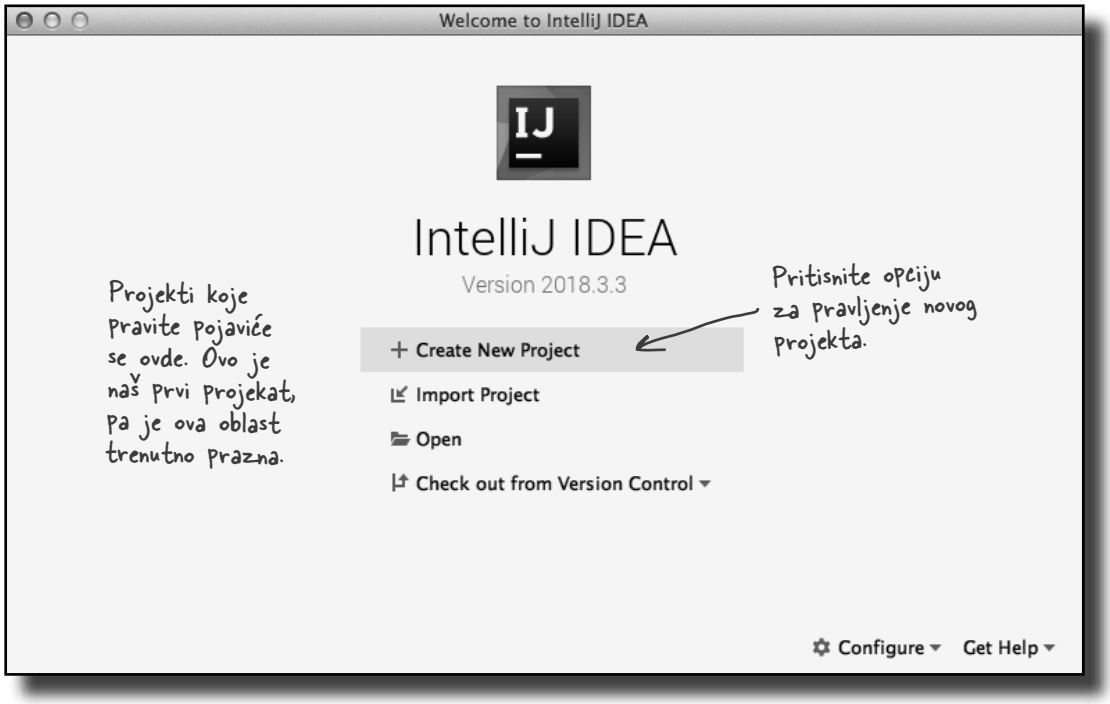
Napravimo osnovnu aplikaciju

Pošto ste podesili okruženje za programiranje, spremni ste za pravljenje svoje prve aplikacije u Kotlinu. Napravićemo veoma jednostavnu aplikaciju koja će prikazivati tekst „Pow!“ u IDE-u.

Kad god pravite novu aplikaciju u okruženju IntelliJ IDEA, treba za nju da napravite nov projekat. Proverite da li je IDE otvoren, i pratite nas.

1. Napravite nov projekat

Uvodni ekran IntelliJ IDEA nudi više opcija za ono što želite da uradite. Mi želimo da napravimo nov projekat, pa pritinsite opciju „Create New Project“.



Pravljenje osnovne aplikacije (nastavak)

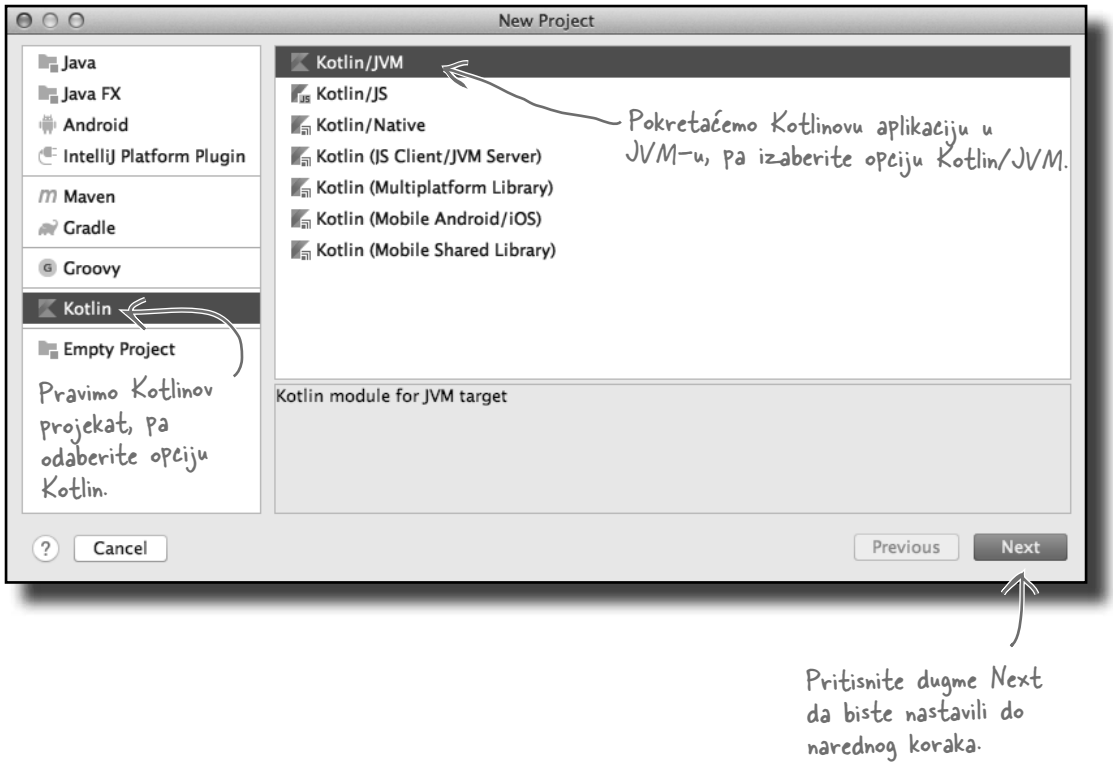
2. Navedite tip projekta

Okruženju IntelliJ IDEA treba da kažete koju vrstu projekta hoćete da napravite.

IntelliJ IDEA dozvoljava da pravite projekte za razne jezike i platforme, kao što su Java i Android. Mi ćemo praviti Kotlinov projekat, pa odaberite opciju „Kotlin“.

Postoje i druge opcije, ali mi ćemo se fokusirati na pravljenjeaplikacija koje se pokreću u JVM-u.

Treba da navedete i ciljnu platformu za Kotlinov projekat. Mi ćemo praviti Kotlinovu aplikaciju kojoj je cilj Javina virtuelna mašina (JVM), pa izaberite opciju Kotlin/JVM. Potom pritisnite dugme Next.



- [x] **Pravljenje aplikacije**
- [ ] **Dodavanje funkcije**
- [ ] **Ažuriranje funkcije**
- [ ] **Korišćenje REPL-a**

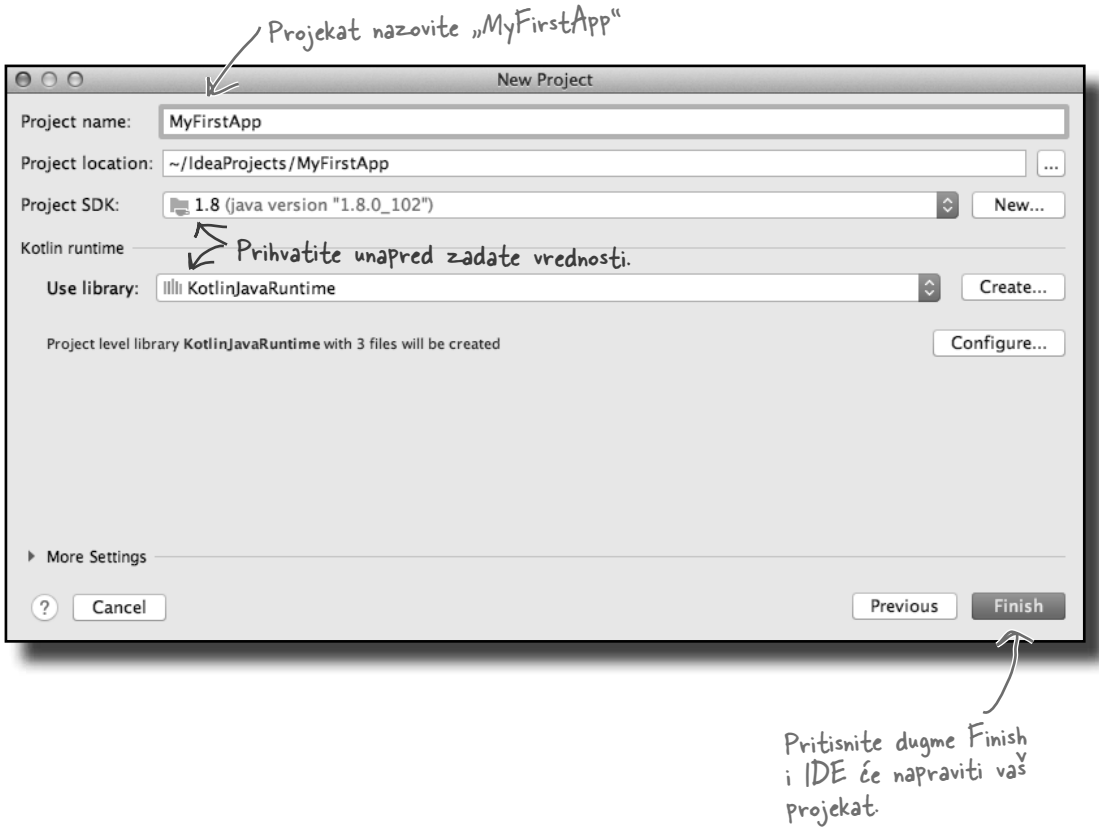
Pravljenje osnovne aplikacije (nastavak)

3. Konfigurisanje projekta

Treba da konfigurišete projekat tako što ćete navesti kako će se on zvati, gde će datoteke biti sačuvane i koje datoteke će projekat koristiti. Ovo uključuje i navođenje verzije Jave koju će koristiti JVM, i biblioteke za izvršavanje Kotlinovog koda.

Nazovite projekat „MyFirstApp“ i prihvatite unapred zadate opcije.

Kada pritisnete dugme Finish, IntelliJ IDEA će napraviti vaš projekat.

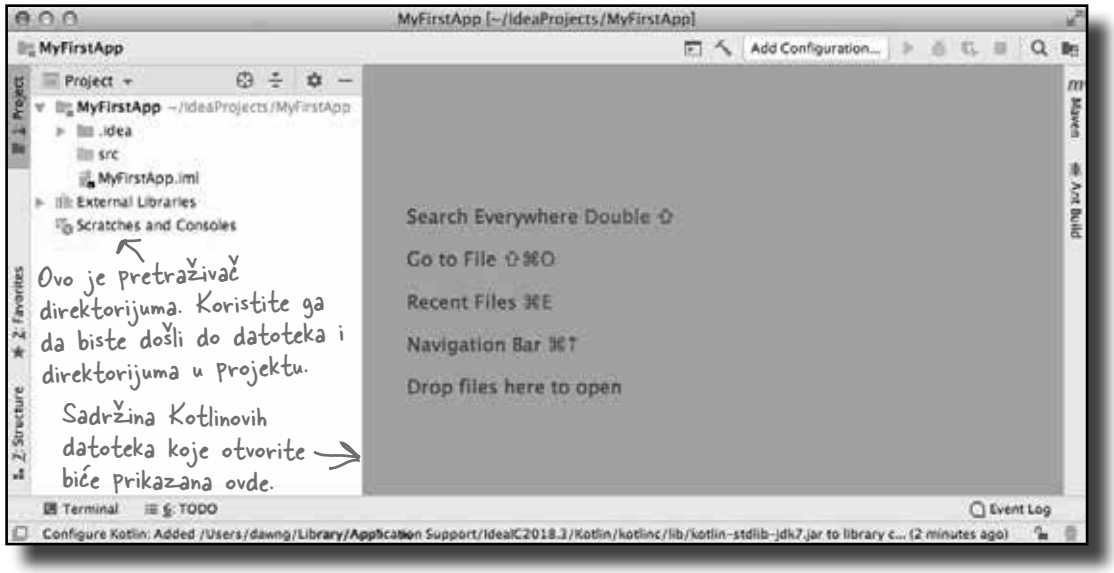


Završili smo ovaj korak, pa ćemo ga štiklirati.

- [x] **Pravljenje aplikacije**
- [ ] Dodavanje funkcije
- [ ] Ažuriranje funkcije
- [ ] Korišćenje REPL-a

Upravo ste napravili svoj prvi Kotlinov projekat

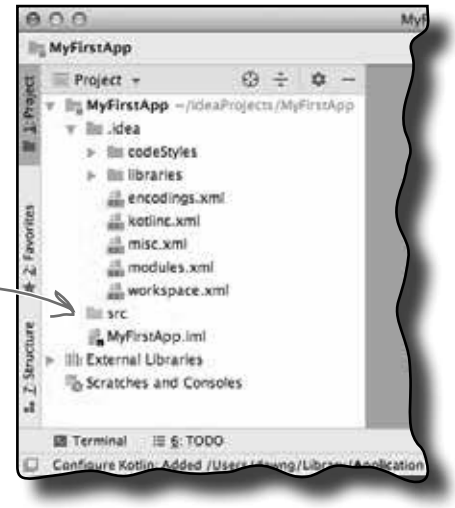
Kada prođete kroz korake pravljenja novog projekta, IntelliJ IDEA umesto vas podešava projekat i prikazuje ga. Evo projekta koji je IDE napravio za nas:



Kao što vidite, projekat sadrži pretraživač pomoću kog ćete dolaziti do datoteka i direktorijuma koji čine projekat. Kada napravite projekat, IntelliJ IDEA umesto vas pravi strukturu direktorijuma.

Struktura direktorijuma sačinjena je od konfiguracionih datoteka koje koristi IDE i spoljnih biblioteka koje će koristiti vaša aplikacija. Ona sadrži i idrektorijum *src*, koji se koristi za čuvanje izvornog koda. Najveći deo vremena u Kotlingradu provešćete radeći sa direktorijumom *src*.

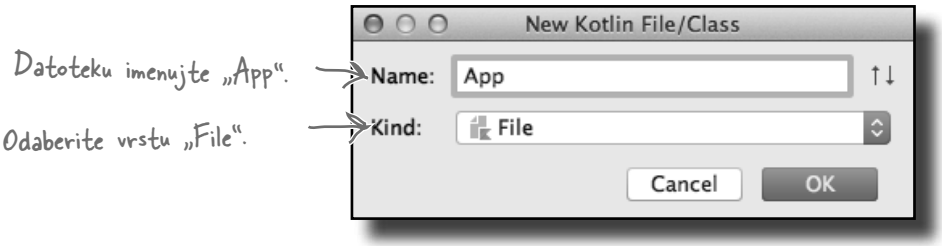
Direktorijum *src* je trenutno prazan jer još uvek nismo dodavali Kotlinove datoteke. To ćemo uraditi u nastavku.



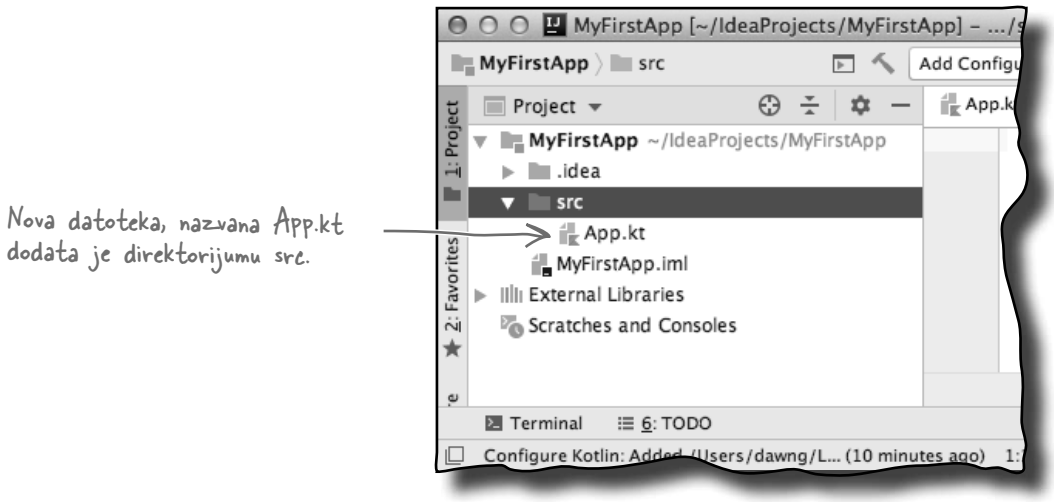
Dodajte projektu novu Kotlinovu datoteku

Pre nego što počnete pisanje Kotlinovog koda, prvo treba da napravite datoteku u koju ćete ga smestiti.

Da biste projektu dodali novu Kotlinovu datoteku, istaknite direktorijum *src* u pretraživaču datoteka IntelliJ IDEA, pa otvorite meni File i odaberite New → Kotlin File/Class. Od vas će se tražiti da unesete ime i tip Kotlinove datoteke koju ćete dodati. Datoteku imenujte „App“, i odaberite File iz liste Kind, ovako:



Kada pritisnete dugme OK, IntelliJ IDEA pravi novu Kotlinovu datoteku nazvanu *App.kt*, i dodaje je direktorijumu *src* u vašem projektu:



Razmotrimo sada kôd koji treba da dodamo datoteci *App.kt* da bi ona nešto radila.

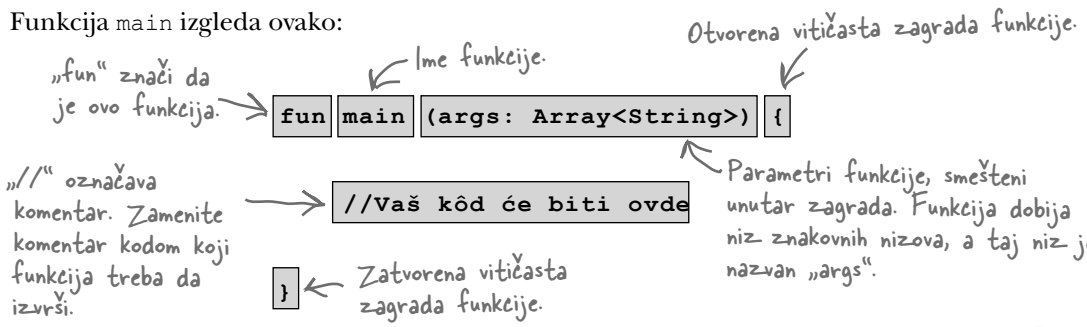
- [x] Pravljenje aplikacije
- [ ] **Dodavanje funkcije**
- [ ] Ažuriranje funkcije
- [ ] Korišćenje REPL-a

Anatomija funkcije main

Napisaćemo Kotlinov kôd koji ispisuje „Pow!“ u IDE-ovom oknu za izlaz. To ćemo postići tako što ćemo datoteci *App.kt* dodati funkciju.

Kad god u Kotlinu pišete aplikaciju, *morate* da joj dodate funkciju `main`, koja započinje aplikaciju. Kada pokrenete kôd, JVM traži tu funkciju i izvršava je.

Funkcija `main` izgleda ovako:

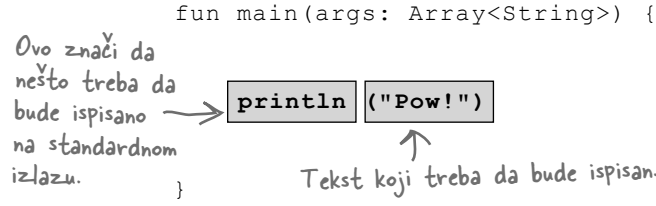


Funkcija počinje rečju **`fun`**, koja Kotlinovom kompajleru govori da je u pitanju funkcija. Rezervisanu reč `fun` ćete koristiti za svaku novu Kotlinovu funkciju koju napišete.

Nakon rezervisane reči `fun` sledi ime funkcije, u ovom slučaju **`main`**. Kada funkciju nazovete `main` to znači da će ona biti automatski izvršena kada pokrenete aplikaciju.

Kôd u zagradama `()` nakon imena funkcije govori kompajleru koje argumente (ako ih ima) funkcija uzima. Ovde kôd `args: Array<String>` navodi da funkcija prihvata bilo koji niz znakovnih nizova (`String`s), i da je taj niz nazvan `args`.

Kôd koji treba da bude izvršen smestićete između vitičastih zagrada `{}` funkcije `main`. Mi želimo da naš kôd ispiše „Pow!“ u IDE-u, a to ćemo postići pomoću ovakvog koda:



`println("Pow!")` ispisuje znakovni niz (engl. *string*), na standardnom izlazu. Pošto kôd izvršavamo u IDE-u, on će ispisati „Pow!“ u njegovom oknu za izlaz.

Pošto ste videli kako izgleda funkcija, dodajmo je našem projektu.

Funkcije main bez parametara

Ako koristite Kotlin 1.2, ili neku stariju verziju, funkcija `main` *mora* da ima naredni oblik da bi pokrenula vašu aplikaciju:

```
fun main(args: Array<String>) {
    //Vaš kôd će biti ovde
}
```

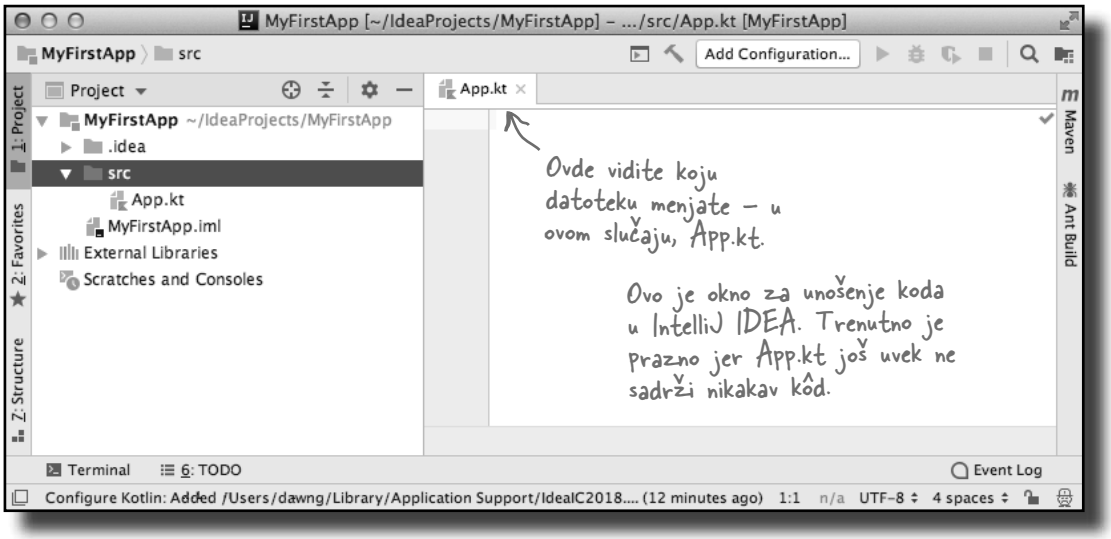
Međutim, od Kotlina1.3, možete da izostavite parametre za `main`, pa će funkcija izgledati ovako:

```
fun main() {
    //Vaš kôd će biti ovde
}
```

U ovoj knjizi ćemo uglavnom koristiti dužu verziju funkcije `main`, jer ona radi u svim verzijama Kotlina.

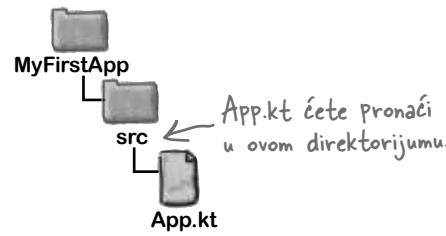
Dodajte funkciju main datoteci App.kt

Da biste funkciju `main` dodali projektu, otvorite datoteku *App.kt* tako što ćete je dvaput pritisnuti u pretraživaču IntelliJ IDEA. Otvoriće se oblast za unošenje koda koju ćete koristiti za pregledanje i menjanje datoteka:



Potom ažurirajte svoju verziju datoteke *App.kt* tkao da bude ista kao ova:

```
fun main(args: Array<String>) {
    println("Pow!")
}
```



Izvršimo kôd da vidimo šta će se desiti.

Glupa pitanja ne postoje

P: Da li funkciju `main` moram da dodam svakoj Kotlinovoj datoteci koju napravim?

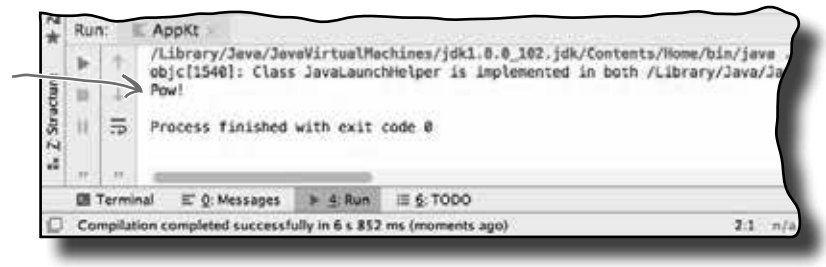
O: Ne. Kotlinova aplikacija može da koristi desetine (ili čak stotine) datoteka, a možda ćete imati samo jednu sa funkcijom `main` — onu koja pokreće izvršavanje aplikacije.

Probna vožnja

Kôd ćete pokrenuti u IntelliJ IDEA-u tako što ćete otvoriti meni Run i odabrati komandu Run. U okviru za dijalog odaberite opciju AppKt. Ovim se gradi projekat i pokreće kôd.

Nakon kraće pauze, trebalo bi da „Pow!“ bude prikazano u oknu za izlaz na dnu IDE-a, ovako:

Evo rezultujućeg teksta u IDE-u.

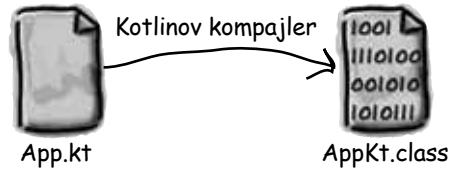


Šta radi komanda Run

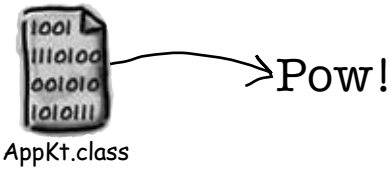
Naš izvorni kôd se kompajlira u JVM-ov binarni kôd jer smo, prilikom pravljenja projekta, izabrali opciju JVM. Da smo odabrali da se on izvršava u nekom drugom okruženju, kompajler bi ga preveo u kôd za to okruženje.

Kada koristite komandu Run, IntelliJ IDEA prolazi kroz par koraka pre nego što vam prikaže rezultat koda:

1. **IDE kompajlira Kotlinov izvorni kôd u JVM binarni kôd.**
Pod pretpostavkom da u kodu nema grešaka, kompajliranjem nastaje jedna ili više datoteka klasa koje mogu da se izvrše u JVM-u. U našem primeru, kompajliranje *App.kt* pravi datoteku klase nazvanu *AppKt.class*.



2. **IDE pokreće JVM i izvršava AppKt.class.**
JVM prevodi binarni kôd *AppKt.class* u nešto što pozadinska platforma razume, a onda ga pokreće. Time se znakovni niz (`String`) „Pow!“ ispisuje u oknu za izlaz u IDE-u.



Pošto znamo da funkcija radi, pogledajmo kako je možemo izmeniti da radi još nešto.

Šta možete da kažete u funkciji main?

Zabava počinje kada ste u funkciji `main` (ili u bilo kojoj drugoj funkciji). Možete reći sve uobičajene stvari koje govorite u većini programskih jezika da bi aplikacija nešto radila.

Moćete reći kodu da:

Nešto uradi (naredbe)

```
var x = 3
val name = "Cormoran"
x = x * 10
print("x is $x.")
//Ovo je komentar
```

Nešto radi opet i opet (petlje)

```
while (x > 20) {
    x = x - 1
    print(" x is now $x.")
}
for (i in 1..10) {
    x = x + 1
    print(" x is now $x.")
}
```

Nešto uradi pod nekim uslovom (grananje)

```
if (x == 20) {
    println(" x must be 20.")
} else {
    println(" x isn't 20.")
}
if (name.equals("Cormoran")) {
    println("$name Strike")
}
```

Razmotrićemo ovo detaljnije na narednim stranama.

Pod lupom: Sintaksa

Evo nekih opštih saveta za sintaksu dok stajete na noge u Kotlinvillu:

★ Jednoredni komentar počinje sa dve kose crte:

```
//Ovo je komentar
```

★ Razmaci uglavnom nisu bitni:

```
x          =          3
```

★ Definišite promenljivu koristeći `var` ili `val`, nakon čega sledi ime promenljive. Koristite `var` za promenljive čiju vrednost ćete želeti da menjate, a `val` za one čija će vrednost ostati ista. O promenljivama ćete više naučiti u poglavlju 2:

```
var x = 100
val serialNo =
    "AS498HG"
```

Prođi opet i opet i opet...

Kotlin ima tri standardne strukture za petlje: `while`, `do-while` i `for`. Zasad ćemo se fokusirati samo na `while`.

Sintaksa petlji `while` relativno je jednostavna. Sve dok je neki uslov istinit, radi sve što je unutar *bloka* petlje. Blok petlje je ograničen vitičastim zagradama, a ono što treba da se ponavlja treba da bude unutar tog bloka.

Ako u bloku petlje imate samo jedan red koda, možete izostaviti vitičaste zagrade.

Ključ za ispravnu petlju `while` jeste njena *provera uslova* (engl. *conditional test*). Provera uslova je izraz koji za rezultat ima logičku (Bulovu) vrednost – nešto što je ili istinito (*true*) ili neistinito (*false*). Na primer, ako kažete nešto poput „Sve dok *imaLiSladoledaUPosudi* ima vrednost *true*, nastavi da sipaš“ imate jasnu logičku proveru. Sladoleda ili ima, ili nema u posudi. Međutim, ako kažete „Sve dok *Fred*, nastavi da sipaš“, nemate pravu proveru. Da bi uslov imao smisla, treba da ga promenite u nešto kao „Sve dok *Fred* jeste gladan, nastavi da sipaš“.

Jednostavne logičke provere

Jednostavnu logičku proveru, Bulov test, možete uraditi tako što ćete proveriti vrednost promenljive koristeći operatore poređenja. U njih spadaju:

< (manje od)

> (veće od)

== (jednakost) ← Za Proveru jednakosti koristite dva znaka jednako, ne jedan.

<= (manje ili jednako)

>= (veće ili jednako)

Obratite pažnju na razliku između operatora dodele (jedan znak jednako) i operatora jednakosti (dva znaka jednako).

Evo nekih primera koda koji koristi logičke provere:

```
var x = 4 //Dodeli 4 promenljivoj x
while (x > 3) {
    //Kôd petlje će se izvršiti jer je x veće od 4
    println(x)
    x = x - 1
}
var z = 27
while (z == 10) {
    //Kôd petlje se neće izvršiti jer z ima vrednost 27
    println(z)
    z = z + 6
}
```

- [x] Pravljenje aplikacije
- [x] Dodavanje funkcije
- [ ] **Ažuriranje funkcije**
- [ ] Korišćenje REPL-a

Upetljani primer

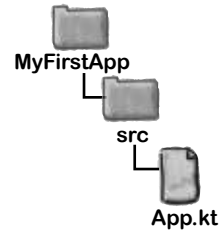
Ažuriraćemo kôd u datoteci *App.kt* novom verzijom funkcije `main`. Izmenićemo funkciju `main` tako da prikazuje poruku pre nego što petlja počne, svaki put kada pravi petlju i kada se petlja završi.

Izmenite datoteku *App.kt* tako da izgleda kao donja (naše izmene su podebljane):

```
fun main(args: Array<String>) {
    println("Pow!")
    var x = 1
    println("Before the loop. x = $x.")
    while (x < 4) {
        println("In the loop. x = $x.")
        x = x + 1
    }
    println("After the loop. x = $x.")
}
```

Obrišite ovaj red jer vam više ne treba.

Ovo ispisuje vrednost x.



Hajde da izvršimo kôd.

Probna vožnja

Pokrenite kôd tako što ćete iz menija Run odabrati komandu Run 'AppKt'. Naredni tekst trebalo bi da se pojavi u oknu za izlaz na dnu IDE-a:

```
Before the loop. x = 1.
In the loop. x = 1.
In the loop. x = 2.
In the loop. x = 3.
After the loop. x = 4.
```

Pošto ste naučili kako rade petlje `while` i logičke provere, osmotrimo naredbu `if`.

print naspram println

Verovatno ste primetili da smo se prebacivali između **`print`** i **`println`**. U čemu je razlika?

`println` umeće *nov* red (`println` je poput skraćenice za „print new line" – ispiši nov red) dok **`print`** sve ispisuje u *istom* redu. Ako hoćete da svaka stvar bude ispisana u svom redu, koristite `println`. Ako hoćete da sve bude zajedno, u istom redu, koristite `print`.

- [x] Pravljenje aplikacije
- [x] Dodavanje funkcije
- [ ] Ažuriranje funkcije
- [ ] Korišćenje REPL-a

Uslovno grananje

Provera `if` slična je logičkoj proveri u petlji `while` osim što, umesto da kažete „***sve dok (while)*** ima sladoleda...“ govorite „***ako (if)*** i dalje ima sladoleda...“

Da biste videli kako to radi, evo koda koji ispisuje znakovni niz ako je jedan broj veći od drugog:

```
fun main(args: Array<String>) {
    val x = 3
    val y = 1
    if (x > y) {
        println("x is greater than y")
    }
    println("This line runs no matter what")
}
```

Ako imate samo jedan red koda u bloku if, možete izostaviti vitičaste zagrade.

Ovaj red se izvršava samo ako je x veće od y.

Gornji kôd izvršava red koji ispisuje „*x* is greater than *y*“ (x je veće od y) samo ako je uslov istinit (`x` jeste veće od `y`). Međutim, bez obzira na to da li je on istinit, red koji ispisuje „This line runs no matter what“ (ovaj red se izvršava bez obzira na sve) biće izvršen. Zavisno od vrednosti `x` i `y`, biće ispisan jedan ili dva iskaza.

Možemo uslovu dodati i `else`, pa možemo reći nešto kao: „ako (*if*) ima sladoleda, nastavi da sipaš, inače (*else*) pojedi sladoled, pa kupi još“.

Evo izmenjene verzije gornjeg koda koja sadrži i `else`:

```
fun main(args: Array<String>) {
    val x = 3
    val y = 1
    if (x > y) {
        println("x is greater than y")
    } else {
        println("x is not greater than y")
    }
    println("This line runs no matter what")
}
```

Ovaj red se izvršava samo ako uslov x > y nije ispunjen.

U većini jezika, to je uglavnom kraj priče o `if`; koristićete ga da izvršite kôd *ako* su uslovi ispunjeni. Kotlin, međutim, ide korak dalje.

- [x] Pravljenje aplikacije
- [x] Dodavanje funkcije
- [ ] Ažuriranje funkcije
- [ ] Korišćenje REPL-a

Korišćenje if za vraćanje vrednosti

U Kotlinu možete da koristite `if` kao **izraz** (engl. *expression*), tako da vraća vrednost. To bi bilo kao da kažete „*ako* u posudi ima sladoleda, vrati jednu vrednost, inače vrati drugačiju vrednost“. Ovaj oblik `if` možete koristiti da biste pisali koncizniji kôd.

Pogledajmo kako to radi tako što ćemo preraditi kôd sa prethodne strane. Ranije smo koristili sledeći kôd da bismo ispisali znakovni niz:

```
if (x > y) {
    println("x is greater than y")
} else {
    println("x is not greater than y")
}
```

Kada if koristite kao izraz, MORATE dodati naredbu else.

Ovo možemo izmeniti koristeći ovakav izraz `if`:

```
println(if (x > y) "x is greater than y" else "x is not greater than y")
```

Kôd:

```
if (x > y) "x is greater than y" else "x is not greater than y"
```

jeste izraz `if`. On prvo proverava uslov za `if`: `x > y`. Ako je taj uslov istinit (*true*), izraz vraća znakovni niz „x is greater than y“. Inače (`else`) uslov je neistinit (*false*), i izraz vraća znakovni niz „x is not greater than y“.

Kôd potom ispisuje vrednost izraza `if` koristeći `println`:

```
println(if (x > y) "x is greater than y" else "x is not greater than y")
```

Ako je x veće od y, kôd ispisuje „x is greater than y“. Ako x nije veće od y, kôd ispisuje „x is not greater than y“.

Tako, ako je `x` veće od `y`, biće ispisano „x is greater than y“ (x je veće od y). Ako nije, biće ispisano „x is not greater than y“ (x nije veće od y).

Kao što vidite, ovakav način korišćenja izraza `if` ima isti rezultat kao kôd sa prethodne strane, ali je sažetiji.

Na narednoj strani dajemo kôd za celu funkciju.

Ažurirajte funkciju main

Ažuriraćemo kôd u datoteci *App.kt* novom verzijom funkcije `main` koja koristi izraz `if`. Zamenite kôd u svojoj verziji *App.kt* tako da odgovara narednom:

```
fun main(args: Array<String>) {
    var x = 1
    println("Before the loop. x = $x.")
    while (x < 4) {
        println("In the loop. x = $x.")
        x = x + 1
    }
    println("After the loop. x = $x.")
    val x = 3
    val y = 1
    println(if (x > y) "x is greater than y" else "x is not greater than y")
    println("This line runs no matter what")
}
```

Obrišite ove redove.

Hajde da isprobamo kôd.

Probna vožnja

Izvršite kôd tako što ćete iz menija Run odabrati komandu Run 'AppKt'. Trebalo bi da se u oknu za izlaz, na dnu IDE-a, pojavi naredni tekst:

```
x is greater than y
This line runs no matter what
```

Pošto ste naučili kako se koristi `if` za uslovno grananje i izraze, okušajte se u narednoj vežbi.

Magneti za kôd

Neko je upotrebio magnete za frižider da bi napisao korisnu novu funkciju **main** koja ispisuje znakovni niz „YabbaDabbaDo“. Nažalost, čudan kuhinjski tornado ispremeštao je magnete. Možete li ponovo da sastavite kôd?

Neće vam biti potrebni svi magneti.

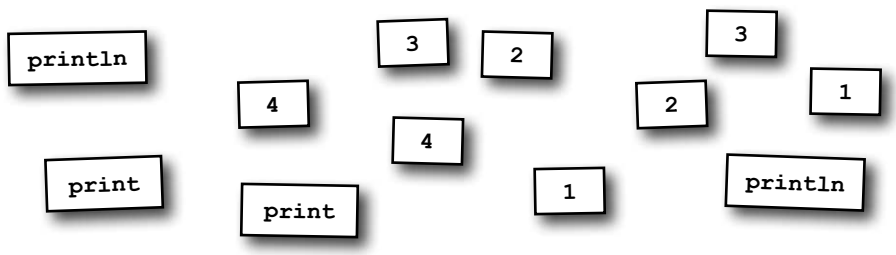
```
fun main(args: Array<String>) {
    var x = 1

    while (x <         ) {

        ..........(if (x ==         ) "Yab" else "Dab")

        ..........("ba")

        x = x + 1
    }
    if (x ==         ) println("Do")
}
```



→ Rešenja na strani 29

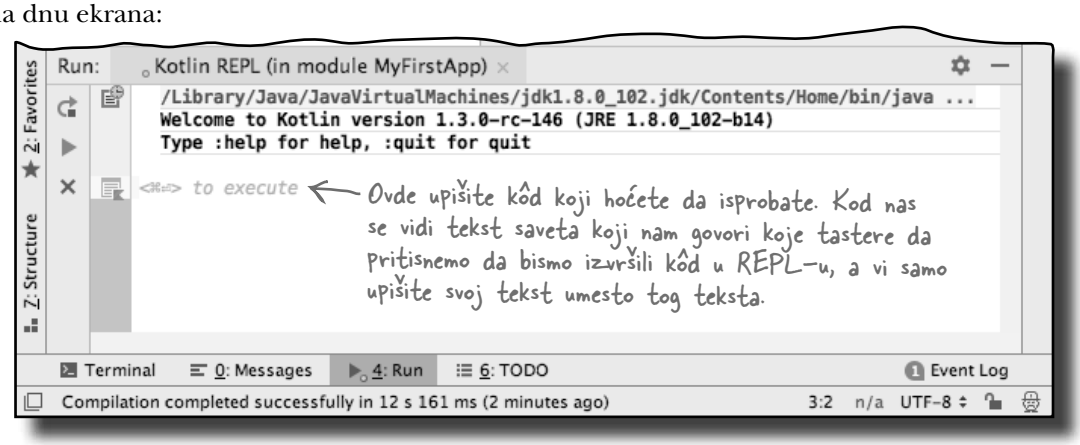
Korišćenje Kotlinovog interaktivnog komandnog okruženja

- [x] Pravljenje aplikacije
- [x] Dodavanje funkcije
- [x] Ažuriranje funkcije
- [ ] **Korišćenje REPL-a**

Pri kraju smo poglavlja, ali pre nego što odemo, treba da vam predstavimo još nešto: Kotlinovo interaktivno komandno okruženje (engl. *interactive shell*), odnosno REPL. REPL omogućava da na brzinu isprobate odlomke koda van glavnog koda.

REPL je skraćenica za Read-Eval-Print Loop (petlja pročitaj-proceni-ispiši) ali niko ga nikada ne zove tako.

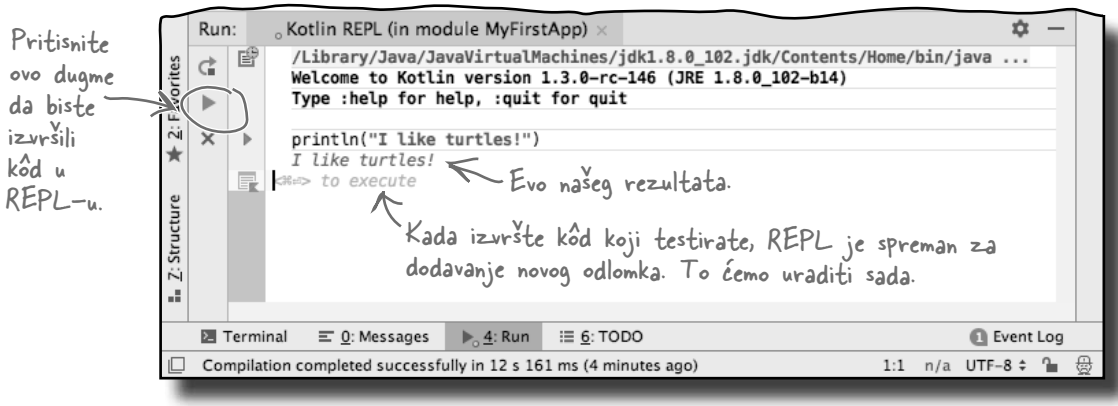
REPL otvarate tako što iz menija Tools u IntelliJ IDEA-u odaberete Kotlin → Kotlin REPL. Otvoriće se novo okno na dnu ekrana:



Da biste koristili REPL, u njegov prozor upišite kôd koji hoćete da isprobate. Kao primer, probajte da dodate sledeće:

```
println("I like turtles!")
```

Kada dodate kôd, izvršite ga pritiskom na veliko zeleno dugme Run na levoj strani prozora. Nakon pauze, trebalo bi da vidite rezultat „I like turtles!“ u prozoru REPL-a:



U REPL možete uneti višeredne odlomke koda

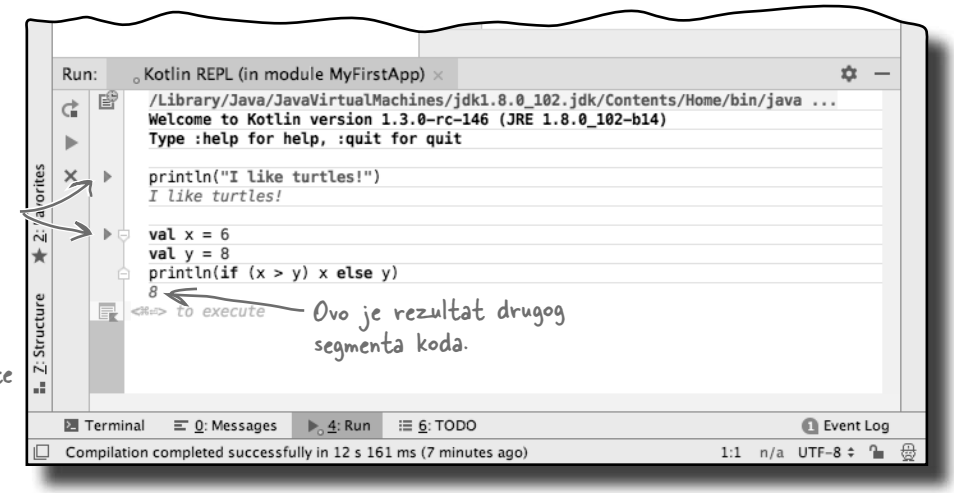
Kao što smo na prethodnoj strani uneli jednoredni odlomak koda u REPL, možete dodati odlomke koji zauzimaju više redova. Kao primer, upišite sledeće redove u prozor REPL-a:

```
val x = 6
val y = 8
println(if (x > y) x else y)
```

Ovo ispisuje broj koji je veći, x ili y.

Kada izvršite kôd, trebalo bi da vidite rezultat `8` u REPL-u:

Ove oznake izgledaju kao male verzije dugmeta za izvršavanje, ali nisu to. One pokazuju koje blokove koda ste izvršili.



Vreme je za vežbe

Pošto ste naučili kako se piše Kotlinov kôd i videli njegovu osnovnu sintaksu, okušajte se u narednim vežbama. Upamtite, ako niste sigurni, u REPL-u možete isprobati bilo koji odlomak koda.

BUDI kompajler

Svaka Kotlinova datoteka na ovoj strani predstavlja potpunu izvornu datoteku. Vaš posao je da se pretvarate da ste kompajler i da utvrdite da li će one biti kompajlirane. Ako neće, kako biste ih popravili?

A

```
fun main(args: Array<String>) {
    var x = 1
    while (x < 10) {
        if (x > 3) {
            println("big x")
        }
    }
}
```

B

```
fun main(args: Array<String>) {
    val x = 10
    while (x > 1) {
        x = x - 1
        if (x < 3) println("small x")
    }
}
```

C

```
fun main(args: Array<String>) {
    var x = 10
    while (x > 1) {
        x = x - 1
        print(if (x < 3) "small x")
    }
}
```

BUDI kompajler Rešenje

Svaka Kotlinova datoteka na ovoj strani predstavlja potpunu izvornu datoteku. Vaš posao je da se pretvarate da ste kompajler i da utvrdite da li će one biti kompajlirane. Ako neće, kako biste ih popravili?

A

```
fun main(args: Array<String>) {
    var x = 1
    while (x < 10) {
        x = x + 1
        if (x > 3) {
            println("big x")
        }
    }
}
```

Ova će biti kompajlirana i izvršiće se bez rezultata, ali ako se programu ne doda red, izvršavaće se večito, u beskonačnoj petlji „while".

B

```
fun main(args: Array<String>) {
    ~~val~~ var x = 10
    while (x > 1) {
        x = x - 1
        if (x < 3) println("small x")
    }
}
```

Ova se neće kompajlirati. x je definisano pomoću val, što znači da mu se vrednost ne može menjati. Zbog toga kôd ne može da ažurira vrednost x unutar petlje „while". Popravite je tako što ćete val zameniti sa var.

C

```
fun main(args: Array<String>) {
    var x = 10
    while (x > 1) {
        x = x - 1
        print(if (x < 3) "small x" else "big x")
    }
}
```

Ova se neće kompajlirati jer koristi izraz if bez naredbe else. Da biste je popravili, dodajte klauzulu else.

Pomešane poruke

U nastavku je dat kratak Kotlinov program. Jedan blok programa nedostaje. Vaš zadatak je da uparite kandidate za blok koda (levo), sa rezultatom koji ćete dobiti ako umetnete taj blok. Neće se koristiti svi redovi rezultata, a neki redovi se mogu koristiti više puta. Linijama povežite kadnidate za kod sa odgovarajućim rezultatom.

```
fun main(args: Array<String>) {
    var x = 0
    var y = 0
    while (x < 5) {
        [                    ]   <- Ovde dolazi kôd kandidata.
        print("$x$y ")
        x = x + 1
    }
}
```

Uparite svakog kandidata sa nekim od mogućih rezultata.

Kandidati:

```
y = x - y
```

```
y = y + x
```

```
y = y + 3
if (y > 4) y = y - 1
```

```
x = x + 2
y = y + x
```

```
if (y < 5) {
    x = x + 1
    if (y < 3) x = x - 1
}
y = y + 3
```

Mogući rezultati:

```
00 11 23 36 410
```

```
00 11 22 33 44
```

```
00 11 21 32 42
```

```
03 15 27 39 411
```

```
22 57
```

```
02 14 25 36 47
```

```
03 26 39 412
```

Pomešane poruke Rešenje

U nastavku je dat kratak Kotlinov program. Jedan blok programa nedostaje. Vaš zadatak je da uparite kandidate za blok koda (levo), sa rezultatom koji ćete dobiti ako umetnete taj blok. Neće se koristiti svi redovi rezultata, a neki redovi se mogu koristiti više puta. Linijama povežite kadnidate za kod sa odgovarajućim rezultatom.

```
fun main(args: Array<String>) {
    var x = 0
    var y = 0
    while (x < 5) {

        print("$x$y ")
        x = x + 1
    }
}
```

Kandidati:

```
y = x - y
```

```
y = y + x
```

```
y = y + 3
if (y > 4) y = y - 1
```

```
x = x + 2
y = y + x
```

```
if (y < 5) {
    x = x + 1
    if (y < 3) x = x - 1
}
y = y + 3
```

Mogući rezultati:

```
00 11 23 36 410
```

```
00 11 22 33 44
```

```
00 11 21 32 42
```

```
03 15 27 39 411
```

```
22 57
```

```
02 14 25 36 47
```

```
03 26 39 412
```

Magneti za kôd Rešenje

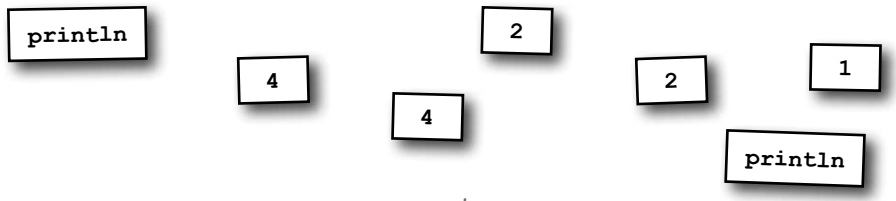
Neko je upotrebio magnete za frižider da bi napisao korisnu novu funkciju **main** koja ispisuje znakovni niz „YabbaDabbaDo“. Nažalost, čudan kuhinjski tornado ispremeštao je magnete. Možete li ponovo da sastavite kôd?

Neće vam biti potrebni svi magneti.

```
fun main(args: Array<String>) {
    var x = 1

    while (x < 3 ) {
        print (if (x == 1 ) "Yab" else "Dab")
        print ("ba")

        x = x + 1
    }
    if (x == 3 ) println("Do")
}
```



Ovi magneti vam nisu trebali.

Kutija sa Kotlinovim alatima

Savladali ste poglavlje 1 i dodali osnovnu Kotlinovu sintaksu svojoj kutiji sa alatom.

Kompletan kôd za poglavlje možete preuzeti sa https://tinyurl.com/HFKotlin.

PRAVO U CENTAR

- Koristite `fun` da biste definisali funkciju.
- Svaka aplikacija mora da ima funkciju nazvanu `main`.
- Koristite `//` da biste označili jednoredni komentar.
- `String` je znakovni niz. Vrednost znakovnog niza označavate tako što znakove postavljate u navodnike.
- Blokovi koda su definisani parom vitičastih zagrada `{ }`.
- Operator dodeljivanja je *jedan* znak jednakosti `=`.
- Operator za jednakost koristi *dva* znaka jednakosti `==`.
- Koristite `var` da biste definisali promenljivu čija se vrednost može menjati.
- Koristite `val` da biste definisali vrednost koja će ostati ista.
- Petlja `while` izvršava sve u svom bloku sve dok je provera uslova istinita (*true*).
- Ako je uslov neistinit (*false*), kôd iz bloka petlje `while` neće biti izvršen, a izvršavanje će preći na kôd neposredno nakon bloka petlje.
- Proveru uslova postavite unutar zagrada `( )`.
- Uslovno grananje dodajte kodu pomoću `if` i `else`. Klauzula `else` nije obavezna.
- Možete koristiti `if` kao izraz tako da vraća neku vrednost. U tom slučaju, klauzula `else` je obavezna.