Kotlin kao programski jezik nove generacije

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Jakovljević Aleksandar, Šerbić Marko, Veljković Marko, Vukadinović Selena
a.jakovljevic96@gmail.com, marko.serbic@gmail.com,
marko.veljko@gmail.com, vukadinovic.selena@gmail.com

6. april 2019

Sažetak

Kotlin je univerzalni programski jezik otvorenog koda, razvijen od strane Džetbrejnsa (engl. *JetBrains*). On je koncizan, bezbedan, pragmatičan i kompatibilan sa Java kodom. Može se koristiti svugde gde se koristi Java, kao što je razvoj serverske strane aplikacija, Android aplikacija i još mnogo toga. Odlično radi sa svim postojećim Java bibliotekama i radnim okvirima, a ima iste performanse kao Java. Sam jezik je nastao u cilju prevazilaženja nedostataka koje je imao programski jezik Java i uspešno opravdao očekivanja mnogih Java programera.

Sadržaj

1 Uvod **2**

2 Nastanak i istorija razvoja **2**

3 Osnovna svrha i namena **3**

4 Osobine jezika **3**

5 Koncepti **4**
5.1 Struktura programa 4
5.2 Deklaracija promenljivih 4
5.3 Struktura petlji 4
5.4 Kontrolne strukture 5
5.5 Sintaksa 5

6 Podržane paradigme **6**

7 Najpoznatija okruženja i karakteristike **7**
7.1 Android 7
7.2 Kolekcije 8
7.3 Spring Boot 8

8 Primer koda i njegovo objašnjene **9**

9 Instalacija i pokretanje **10**
9.1 Instalacija 10
9.2 Pokretanje 10

10 Zaključak **11**

Literatura **11**

A Dodatak: Skraćenice **12**

1 Uvod

U radu ćemo prikazati osnovne karakteristike programskog jezika Kotlin, videćemo šta je to što ga izdvaja od drugih jezika i po čemu je specifičan. Najpre ćemo proći kroz nastanak i istorijski razvoj programskog jezika, kao i uticaje drugih jezika na njega. Objasnićemo najvažnije osobine, paradigme koje podržava i osnovne namene samog jezika. Uz osobine i namene, neizostavni deo su svakako okruženja (engl. *framework*) za korišćenje jezika, približićemo čitaocu neke od najpoznatijih okruženja kao što su Android, kolekcije i *Spring Boot*. Daćemo detaljno uputstvo za instalaciju i pokretanje programskog jezika preko komandne linije. Na kraju ćete videti primer jednog kratkog koda uz detaljno objašnjenje u kojem se može videti samo deo konciznosti i snage ovog jezika. Autori ovog rada pretpostavljaju da su čitaoci upoznati sa programskim jezikom Java.

2 Nastanak i istorija razvoja

Jezik je nastao 2011. godine, kao jezik za rad na Java virtuelnoj mašini (JVM). Kako je rekao vođa projekta u kompaniji Džetbrejns, Dmitri Jemerov:

"Skup karakteristika drugih jezika ne zadovoljava naše potrebe."

Jedino je Skala (engl. *Scala*) bila veoma blizu, ali problem sa njom je bilo sporije prevođenje u odnosu na Javu. Prvi cilj razvoja Kotlina bio je jezik moćan kao Skala, ali brz kao Java. U februaru 2012. Džetbrejns je postavio otvoreni kod sa Apač 2 (engl. *Apache 2*) licencom. Četiri godine kasnije, 15. februara 2016., izlazi prva zvanična verzija, **Kotlin 1.0**. Još jedan bitan korak u razvoju jezika jeste Gugl (engl. *Google*) I/O konferencija, održana maja 2017. na kojoj je Kotlin proglašen zvaničnim jezikom za razvoj Android aplikacija uz punu podršku Gugla. U verziji 1.1 omogućeno je da pored Java koda, i kod pisan u Javaskriptu (engl. *JavaScript*) prevodi i pokreće iz veb pregledača. Od verzije 1.2 moguće je deljenje koda izmedju JVM i Javaskript platformi, dok od verzije 1.3 korutine postaju stabilno svojstvo jezika za pisanje neblokirajućeg koda. Trenutno je u toku razvoj multiplatformskog svojstva, tako da bi se u budućnosti Kotlin mogao koristiti i za razvoj klijentskih strana aplikacija. Do sada su objavljene i *Kotlin/Native*, kao beta verzija, za kompilaciju koda direktno u mašinske instrukcije, kao i Ktor v1.0, veb radni okvir (engl. *web framework*) za Kotlin, koji je namenjen za pravljenje asinhronih servera u povezanim sistemima.

Kao što je već vise nego očigledno, Kotlin je nastao po uzoru na Javu, tako da su ta dva jezika dosta povezana. Sa jedne strane tokom pisanja koda u jeziku Kotlin, moguće je dodavati i Java kod, kao i uključivanje postojećih Java biblioteka. Zahvaljujući bliskosti sa Javom, sve klase Java koda mogu se instancirati i referisati u Kotlin kodu bez upotrebe posebne semantike, što važi i obrnuto. Posledica ovoga jeste mogućnost pripajanja koda pisanog u Kotlinu u već postojeće projekte. Razlikuju se u tome što Kotlin ima jednostavniju sintaksu, pa je moguće uraditi više sa manje koda. Kod je čitljiviji i lakši za razumevanje. [2] [7] Prethodna istraživanja su pokazala da je u nekim slučajevima vreme izvršavanja kraće od vremena izvršavanja Java koda, jer između ostalog Kotlin podržava umetanje (engl. *inline*) čitavog tela funkcije, kao i da program koji je napisan u Kotlinu zauzima bar onoliko memorije koliko bi zauzimao program napisan u Javi.

Vreme prevođenja je uglavnom sporije nego što je to u slučaju sa Javom. [10]

3 Osnovna svrha i namena

Osnovna svrha programskog jezika Kotlin je da obezbedi koncizniju, produktivniju i bezbedniju alternativu Javi, takvu da se može primeniti svuda gde se danas koristi Java. Java je veoma popularan jezik i koristi se u raznim okruženjima. U većini slučajeva, korišćenje Kotlina može pomoći razvijaocima da postignu isti rezultat uz manju količinu koda i manje problema na putu do rešenja. Najčešće primene Kotlina su pisanje aplikacija za servere i pisanje aplikacija za Android uređaje, mada se može koristiti za pisanje aplikacija za uređaje koji radi na operativnom sistemu iOS uz korišćenje alata *Intel Multi-OS Engine*, i za pravljenje desktop aplikacija uz TornadoFX [11] i JavaFX okruženja. Kotlin se može prevesti u Javaskript, pa se tako može pokretati u web pregledaču, ali je ovaj koncept još uvek samo prototip. [6] [4]

4 Osobine jezika

Već smo napomenuli da je Kotlin sličan programskom jeziku Java, ali važno je istaći neke od osobina koje sam jezik poseduje.[6] [4] [9] [7]

1. **Statički je tipiziran.**
   Tip svakog izraza je poznat u vreme prevođenja, ali Kotlin ne zahteva da se eksplicitno naglasi u kodu. U većini slučajeva tip promenljive se može zaključiti iz konteksta.
2. **Funkcionalan i objektno-orijentisan.**
   Kotlin je objektno-orijentisan jezik. Nije čisto funckionalan jezik, ali poput mnogih modernih jezika koristi mnoge koncepte funkcionalnog programiranja i pruža veliku podršku funkcionalnom programiranju.
3. **Besplatan i otvorenog koda.**
   Prevodilac, biblioteke i svi dodatni alati za Kotlin su besplatni i otvorenog koda.
4. **Bezbedan.**
   Moramo eksplicitno navesti da objekat može imati null vrednost i pre korišćenja objekta moramo proveriti koju vrednost on zapravo ima. Kotlin ovo proverava tokom prevođenja, kako bi izbegao moguće probleme sa izuzecima. Ova osobina može programeru uštedeti mnogo vremena tokom debagovanja.
5. **Koristi funkcije za nadograđivanje.**
   Možemo proširiti neku klasu dodatnim funckionalnostima, bez promene postojećeg koda.
6. **Kompatibilan sa Javom.**
   Moguće je napraviti "mešovite" projekte, koji bi imali u sebi datoteke pisane i u Javi, i u Kotlinu.
7. **Odlična podrška razvojnih okruženja od prvog dana.**
   Pored Kotlina, kompanija DžetBrejns je razvila i svima dobro poznat IDE, Intelidžej IDEA (engl. *IntelliJ IDEA*), u kojem je Kotlin imao punu podršku od prvog dana objavljivanja. Pored ovoga, razvijen je Kotlin dodatak (engl. plugin) za Eklips(engl. *Eclipse IDE*).

Podrška unutar razvojnih okruženja omogućila je brže učenja jezika, pravljenje manjeg broja grešaka i pisanje kvalitetnijeg koda.

5 Koncepti

Kao što je napomenuto više puta do sada, Kotlin je sličan Javi, pa je samim tim i veliki broj koncepta preuzet iz Jave. Ova lista koncepta nije potpuna i potpunu listu možete potražiti na zvaničnom sajtu jezika. [5]

5.1 Struktura programa

Svaki program sadrži funkciju **main**, koja predstavlja glavnu nit izvršavanja. Uključivanje dinamičkih bibloteka vrši se pomoću ključne reči **import**. Ne postoji neka velika razlika u odnosu na Javu kada je u pitanju sama struktura programa.

Primer 1: Struktura programa

```
import java.util.Scanner

fun main(args: Array<String>) {
    // Isto kao kod Jave, uz drugaciju sintaksu.
    val reader = Scanner(System.`in`)
    var integer:Int = reader.nextInt()
    println("You entered: $integer")
}
```

5.2 Deklaracija promenljivih

Postoje dva različita načina za definisanje promenljivih: pomoću ključne reči **val** koja označava konstantnu promenljivu, onu koja se neće menjati i uz ključnu reč **var** koja označava promenljivu čija vrednost u budućnosti može biti promenjena.

Primer 2: Deklarisanje promenljivih

```
    var a: String = "initial"
    println(a)
    val b: Int = 1
    val c = 3
```

5.3 Struktura petlji

Kao i u većini programskih jezika, i u Kotlinu postoje tri osnovne petlje: **while**, **do-while** i **for**. **For** se može koristiti kao standardna **for** petlja ili kao **foreach** petlja kao što je prikazano u primeru 3.

Primer 3: Strukture petlji

```
    var cakesEaten = 0, cakesBaked = 0
    while (cakesEaten < 5) {
        println("Eat a Cake")
        cakesEaten ++
    }
```

```
    do {
        println("Bake a Cake")
        cakesBaked++
    } while (cakesBaked < cakesEaten)
    val cakes = listOf("carrot", "cheese",
                       "chocolate")
    for (cake in cakes) {
        println("Yummy,it's a $cake cake!")
    }
```

5.4 Kontrolne strukture

Jedna od novina koja je uvedena u Kotlinu jeste naredba **when**, koja se koristi na sličan način kao **switch** u Javi. Pored ovoga, još jedna novost je da se **if** ponaša kao izraz, odnosno vraća povratnu vrednost, pa se istovremeno koristi i kao zamena za standardni ternarni operator **(:?)**.

Primer 4: Kontorlne strukture

```
fun max(a: Int, b: Int) = if (a > b) a else b

fun cases(obj: Any) {
    when (obj) {
        1 -> println("One")
        "Hello" -> println("Greeting")
        is Long -> println("Long")
        !is String -> println("Not")
        else -> println("Unknown")
    }
}

fun whenAssign(obj: Any): Any {
    val result = when (obj) {
        1 -> "one"
        "Hello" -> 1
        is Long -> false
        else -> 42
    }
    return result
}
```

5.5 Sintaksa

Funkcije se definišu uz korišćenje ključne reči **fun**, a mogu se ponašati i kao izrazi, odnosno nije obavezno navođenje ključne reči **return**.

Primer 5: Definisanje funkcije

```
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun sum(a: Int, b: Int) = a + b
```

Komentari su standarni kao u Javi, '//' za jednolinijske i '/* */' za višelinijske.

Primer 6: Komentari

```
// This is an end-of-line comment

/* This is a block comment
   on multiple lines. */
```

Još jedna od mogućnosti koju pruža Kotlin je korišćenje intervala, odnosno generisanje niza brojeva iz nekog intervala, uz eventualno dodavanje koraka ili brojanja unazad.

Primer 7: Korišćenje intervala

```
val x = 10, y = 9
if (x in 1..y+1) {
    println("fits␣in␣range")
}
```

Kotlin pruža podršku u pisanju "čistog" API-ja. Na ovaj način čitaocu koda je mnogo jasnije čemu služi napisani kod. Da bi se postiglo to, kod treba da izgleda "čisto" sa minimalnom ceremonijom i bez nepotrebne sintakse. Tabela 1 pokazuje na koji način se upotrebljavaju neke odlike jezike pomoću kojih se smanjuje šum u kodu.

Tabela 1: Primer korišćenja čiste sintakse.

regularna sintaksa	čista sintaksa	iskorišćena odlika
StringUtil.capitalize(s)	s.capitalize()	funkcije proširenja
set.add(2)	set += 2	preopterećenje operatora
map.get("key")	map["key"]	konvencija za *get* metodu
file.use(f -> f.read())	file.use it.read()	lambda izvan zagrada

sb.append("yes")
sb.append("no")
	with (sb) {
	append("yes")
	append("no")
	}
		lambda sa resiverom

6 Podržane paradigme

Navodeći osobine jezika, spomenuli smo da Kotlin podržava objektno-orijentisanu paradigmu i funkcionalnu paradigmu. 4 Kod funkcionalne paradigme, funkcije su građani višeg reda, pa su omogućene i apstrakcije višeg reda. Osnovni skup karakteristika kojima Kotlin podržava funkcionalno programiranje čine:

- Funkcionalni tipovi koji omogućavaju prosleđivanje funkcije kao argument druge funkcije.
- Lambda izrazi.
- Klase podataka (engl. *Data classes*). 8 [5]
  - Omogućava konciznu sintaksu za kreiranje konstantnih objekata.

  - Generiše i oslobađa nas pisanja preopširnih *get*, *set* i drugih metoda iz Jave.

- Obiman skup API-ja u standardnoj biblioteci za rad nad objektima i kolekcijama u funkcionalnom stilu.

Primer 8: Data klasa

```
data class User(val name: String = "")
```

Ovo ne znači da Kotlin forsira funkcionalni stil programiranja. Ukoliko želimo uvek možemo da koristimo promenljive koje menjaju svoje vrednosti i da imamo sporedne efekte. Korišćenje hijerarhija klasa i interfejsa identično je kao u Javi. Pored navedenih, Koltin takođe podržava reaktivnu i konkurentnu paradigmu. [3] [9]

7 Najpoznatija okruženja i karakteristike

Programski okvir je univerzalno softversko rešenje koje se može koristiti nebrojeno puta. Omogućava određenu funkcionalnost kao deo veće softverske platforme, a to je da olakša razvoj aplikacija, proizvoda i rešenja.

7.1 Android

Kako je Gugl 2017. godine na svojoj godišnjoj Gugl I/O konferenciji objavio punu podršku za Kotlin i time ga proglasio zvaničnim jezikom za razvoj Android aplikacija, pozabavićemo se okvirima koji pripadaju ovoj kategoriji.

- **JVM-based GUI**
  Većina radnih okvira grafičkog korisničkog interfejsa (engl. *GUI*) koji se zasnivaju na Java virtuelnoj mašini (engl. *JVM*) imaju jednu zajedničku osobinu, pokreću posebnu nit koja je odgovorna za ažuriranje stanja korisničkog interfejsa (engl. *UI*) aplikacije. Mehanizam zadužen za upravljanje nitima, obavlja poslove asinhrono u pozadini, ali se mora prebaciti na UI nit svaki put kada ažurira stanje UI-a. [12]

- **JUnit**
  JUnit radni okvir se koristi za testiranje pojedinačnih delova koda. Jedna od mogućnosti ovog okvira je pisanje testova koji proveravaju ispravnost koda koji se asinhrono izvršava u pozadini, dok mehanizam obavlja druge zadatke na glavnoj niti. Uz korišćenje Kotlinove *Mockito* biblioteke, testiranje asinhronog koda se obavlja sa lakoćom. [12]

- **DBFlow**
  DBFlow je deo SQLite sistema za upravljanje relacionim bazama podataka, koji olakšava interakciju sa njima. Ovo se postiže, kao i u većini Android radnih okvira, korišćenjem anotacija. [5]

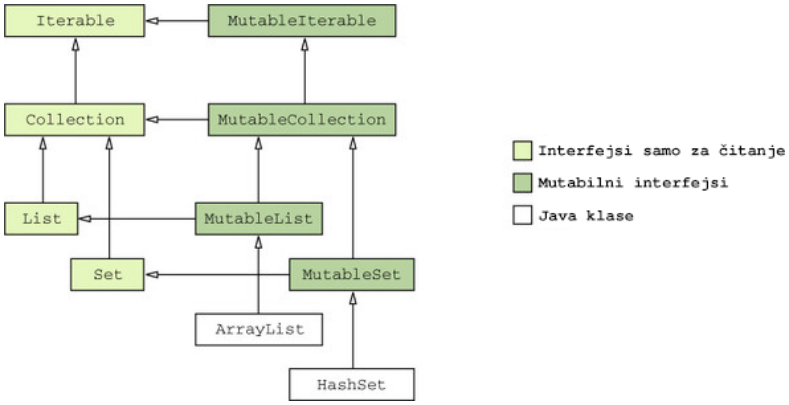
Primer 9: Deklarisanje tabele

```
@Table(name="users", database =
        AppDatabase::class)
class User : BaseModel() {
    @PrimaryKey(autoincrement = true)
    @Column(name = "id")
```

```
    var id: Long = 0
    @Column (name = "name")
    var name: String? = null
}
```

7.2 Kolekcije

Radni okvir kolekcija je koristan kada želimo da radimo sa kolekcijama elemenata. Najkorišćenije kolekcije, koje možete i videti na slici 1, su liste, mape i skupovi. Okvir kolekcija u Kotlinu je mnogo bolji od onog u Javi, zbog mogućnosti funkcionalnog programiranja, čime naš kod postaje koncizniji i lakši za razumevanje. Neke od osnovnih funkcija koje se nalaze u ovom okviru su spajanje i razdvajanje kolekcija, sortiranje po jednom ili više parametara, funkcije višeg reda (funkcije koje primaju druge funkcije kao parametre, ili čiji je rezultat funkcija) kao što su mapiranje, redukovanje i filtriranje. [8]



Slika 1: Interfejsi i klase kolekcija

7.3 Spring Boot

Spring Boot predstavlja naslednika Spring veb okvira. Za razliku od Springa, Spring Boot preferira konvenciju pre konfiguracije, čime se uklanja velika količina bojlerplejt (engl. *Boilerplate*) koda. Bojlerplejt je tekst koji kada se jednom napiše, može koristiti u različitim situacijama uz minimalne izmene sadržaja. Okvir se može koristiti za pravljenje mikro servisa. Olakšava brzo oblikovanje proizvoda i uključuje karaktirestike koje dovode aplikaciju do proizvodnje bez po muke. [1]

8 Primer koda i njegovo objašnjene

Primer 10: Funkcija koja pravi listu url adresa

```
override fun doInBackground
            (vararg params: Void?): Void? {
    val url = "http://poincare.matf.bg.ac.rs"+
              "/~kmiljan/raspored/sve/"
    val indeksiStranica = arrayListOf("index.html")
    for (i in 1..41){
        var indeks = "form_0"
        indeks += (if(i<10) "0" else "")
               + i.toString()+".html"
        indeksiStranica.add(indeks)
    }
    for (indeks in indeksiStranica)
        try {
            println(url+indeks)
        } catch (e: Exception) {
            e.printStackTrace()
        }
    return null
}
```

Posmatrajmo prevazilaženje (eng. *override*) funkcije doInBackground, koja pripada klasi AsyncTask. Ona prihvata argumente tipa Void?, što znači da argument ne mora postojati ili može biti null. Isto važi i za povratni tip funkcije. Vidimo da jezik nije strogo tipiziran, pa samim tim prilikom definisanja promenljivih ne navodimo njihov tip, već se on razrešava prilikom prevođenja. Umesto tipa pre imena promenljive navodi se ključna reč val ili var u zavisnosti da li hoćemo da deklarišemo promenljivu koja se ne može menjati(val) ili čiju ćemo vrednost u budućnosti želeti da promenimo(var). Unutar promenljive url smeštamo nisku koji označava početak svih url adresa rasporeda časova na MATF-u. indeksiStranica predstavlja promenljivu listu, koja sadrži krajeve svih url stranica za pomenuti raspored, u kojoj je na početku smešten samo jedan element koji predstavlja kraj prve url stranice.

Zatim imamo for petlju u kojoj promenljiva i iterira kroz raspon brojeva od 1 do 41, i u svakoj iteraciji proverava veličinu broja i. Ako je manji od 10 na promenljivu indeks se dodaje niska "0" zbog kompatibilnosti sa originalnim url adresama. Ovde možemo primetiti interesantan način implementiranja ternarnog operatora, umesto klasične sintakse (?:), kreatori jezika su se odlučili da omoguće korišćenje if-then-else naredbe unutar već postojeće naredbe i time dobiju na konciznosti koda. Na kraju svake iteracije, dodajemo trenutni element unutar liste indeksiStranica.

Druga for petlja zapravo predstavlja foreach petlju, u kojoj navodimo da želimo da iteriramo kroz listu indeksiStranica i pristupimo njenim elementima. Unutar for petlje je dodat try-catch blok samo radi demonstarcije. Vidimo da je njegova sintaksa ostala ista kao i u Javi. Na kraju unutar try dela ispisujemo sve url adrese na standardni izlaz, liniju po liniju. U implementaciji funkcije doInBackground njena povratna vrednost je null, a kako se ta vrednost prenosi daljim funkcijama unutar klase AsyncTask, ovu vrednost nismo menjali.

9 Instalacija i pokretanje

Poput koda napisanog u Javi, i kod napisan u Kotlinu moguće je prevesti koristeći komandnu liniju. Za ovakav poduhvat, potrebno je instalisati prevodilac za programski jezik Kotlin koji se naziva kotlinc. Prevodilac se može instalisati na više načina u zavisnosti od operativnog sistemima i u daljem tekstu sledi opis koraka za instalisanje i pokretanje programa napisanih u Kotlinu za Linuks (engl. *Linux*) i Vindouz (engl. *Windows*) operativne sisteme. Potrebno je napomenuti da je za pokretanje kotlinc prevodioca nephodno imati instalisanu Javu. Osim prevođenja preko komandne linije, prevodilac za Kotlin dolazi u sklopu dva razvojna okruženja: Intelidžej i Eklips. U daljem tekstu, bavićemo se pokretanjem programa putem komandne linije, dok čitaocu ostavljamo da se samostalno upozna sa radom u prethodno navedenim razvojnim okruženjima. [5]

9.1 Instalacija

Jedan od načina za instalisanje prevodioca za programski jezik Kotlin jeste ručno instalisanje koje zahteva od korisnika da preuzme najnoviju verziju prevodioca Githaba (engl. *GitHub*). Nakon preuzimanja sadržaja, potrebno je otpakovati sadržaj i opciono se može podesiti sistemska putanja za prevodilac kako bi postojala mogućnost pokretanja prevodioca putem komandne linije jednostavnom komandom kotlinc.

Pored ručnog instalisanja, moguće je instalisati prevodilac korišćenjem SDKMAN alata. Za instalisanje alata potrebno je izvršiti narednu komandu:

```
$ curl -s https://get.sdkman.io | bash
```

Nakon preuzimanja alata, jednostavnom komandom vršimo instalisanje prevodioca:

```
$ sdk install kotlin
```

Pored do sada navedenih načina, Linuksove distribucije Ubuntu 16.04 i novije, podržavaju i instalisanje preko Snepkraft (engl. *SnapCraft*) alata izvršavanjem naredne komande:

```
$ sudo snap install --classic kotlin
```

9.2 Pokretanje

Da bi se program pokrenuo, potrebno ga je prvo prevesti. Prevođenje vršimo pozivanjem naredbe iz komandne linije:

```
$ kotlinc <izvorni>.kt -include-runtime -d <ime>.jar
```

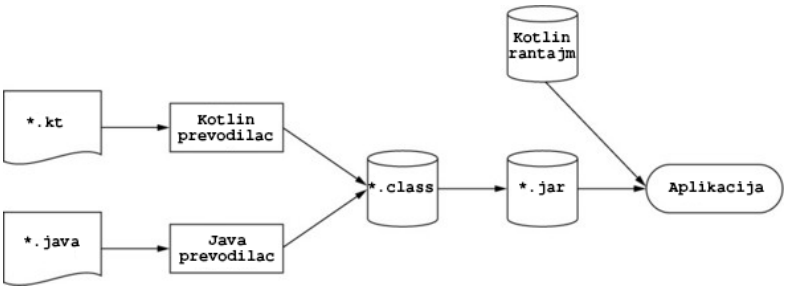
U navedenoj komandi, -include-runtime opcija uključuje Kotlinovu biblioteku izvršavanja u dobijeni .jar, dok opcijom –d navodimo odredište za generisane datoteke klasa. Celokupan postupak prevođenja možete videti na priloženoj slici.2 Nakon prevođenja, izvršavanje programa se poziva naredbom:

```
$ kotlin <ime>.jar
```

ili

```
$ java -jar <ime>.jar
```

Ukoliko želimo da prevedemo biblioteku koju će da koriste druge aplikacije napisane u Kotlinu, dovoljno je izbaciti –include-runtime opciju iz komande 9.2. Kotlin može da se izvršava i u interaktivnom okruženju, REPL, navođenjem samo imena prevodioca kao argument komandne linije. [5] [9]



Slika 2: Postupak prevođenja izvornog koda u izvršni program.

10 Zaključak

Osnovna ideja rada bila je uvod u programski jezik Kotlin, prikaz njegovih osnovnih karakteristika i prednosti u odnosu na druge jezike, sa posebnim osvrtom na Javu. Videli smo da je to pre svega statička tipiziranost i netipiziranost, konciznost, sigurnost, pragmatičnost i robusnost. Ove osobine dovode do svakodnevnog rasta Kotlin zajednice, čime se proširuje opšta slika o jeziku i ubrzava proces unapređivanja performansi. Zbog svega nevedenog, očekuje se da Kotlin preuzme vodeću ulogu u razvoju Android aplikacija i time postane jedan od glavnih programskih jezika u svetu programiranja.

Literatura

[1] A. Belagali, H. Trivedi, and A. Chordiya. *Kotlin Blueprints: A practical guide to building industry-grade web, mobile, and desktop applications in Kotlin using frameworks such as Spring Boot and Node.js*. Packt Publishing, 2017.

[2] Roman Belov. JetBrains Blog, 2018. on-line at: https://blog.jetbrains.com/kotlin/2018/10/kotlin-1-3/.

[3] R. Chakraborty. *Reactive Programming in Kotlin*. Packt Publishing, 2017.

[4] D. Jemerov and S. Isakova. *Kotlin in Action*. Manning Publications Company, 2016.

[5] Kotlin. Kotlin Programming Language, 2019. on-line at: https://kotlinlang.org/.

[6] A. Leiva and Lean Publishing. *Kotlin for Android Developers: Learn Kotlin the Easy Way While Developing an Android App*. CreateSpace Independent Publishing Platform, 2016.

[7] Mrs.T.Kamaleswari Mrs.J.ArockiaJeyanthi. KOTLIN - A New Programming Language for the Modern Needs. *International Journal of Science, Engineering and Management (IJSEM)*, 2(62):271–275, December 2017.

[8] A.S. Roy and R. Karanpuria. *Kotlin Programming Cookbook: Explore More Than 100 Recipes That Show How to Build Robust Mobile and Web Applications with Kotlin, Spring Boot, and Android*. Packt Publishing, 2018.

[9] S. Samuel and S. Bocutiu. *Programming Kotlin*. Packt Publishing, 2017.

[10] Patrik Schwermer. Performance evaluation of kotlin and java on android runtime. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2018.

[11] Edvin Syse. TornadoFX. on-line at: https://github.com/edvin/tornadofx.

[12] S. Urbanowicz. *Kotlin Standard Library Cookbook: Master the powerful Kotlin standard library through practical code examples*. Packt Publishing, 2018.

A Dodatak: Skraćenice

- API - Application programming interface
- GUI - Graphic user interface
- IDE - Integrated development environment
- I/O - Input - output
- JAR - Java archive
- JVM - Java virtual machine
- OS - Operating system
- REPL - Read-Eval-Print-Loop
- SDKMAN - Software Development Kit Manager
- UI - User interface