OSNOVE RAZVOJA WEB I MOBILNIH APLIKACIJA

LV 5: Uvod u programski jezik Kotlin

Osijek, 2023.

1. PRIPREMA

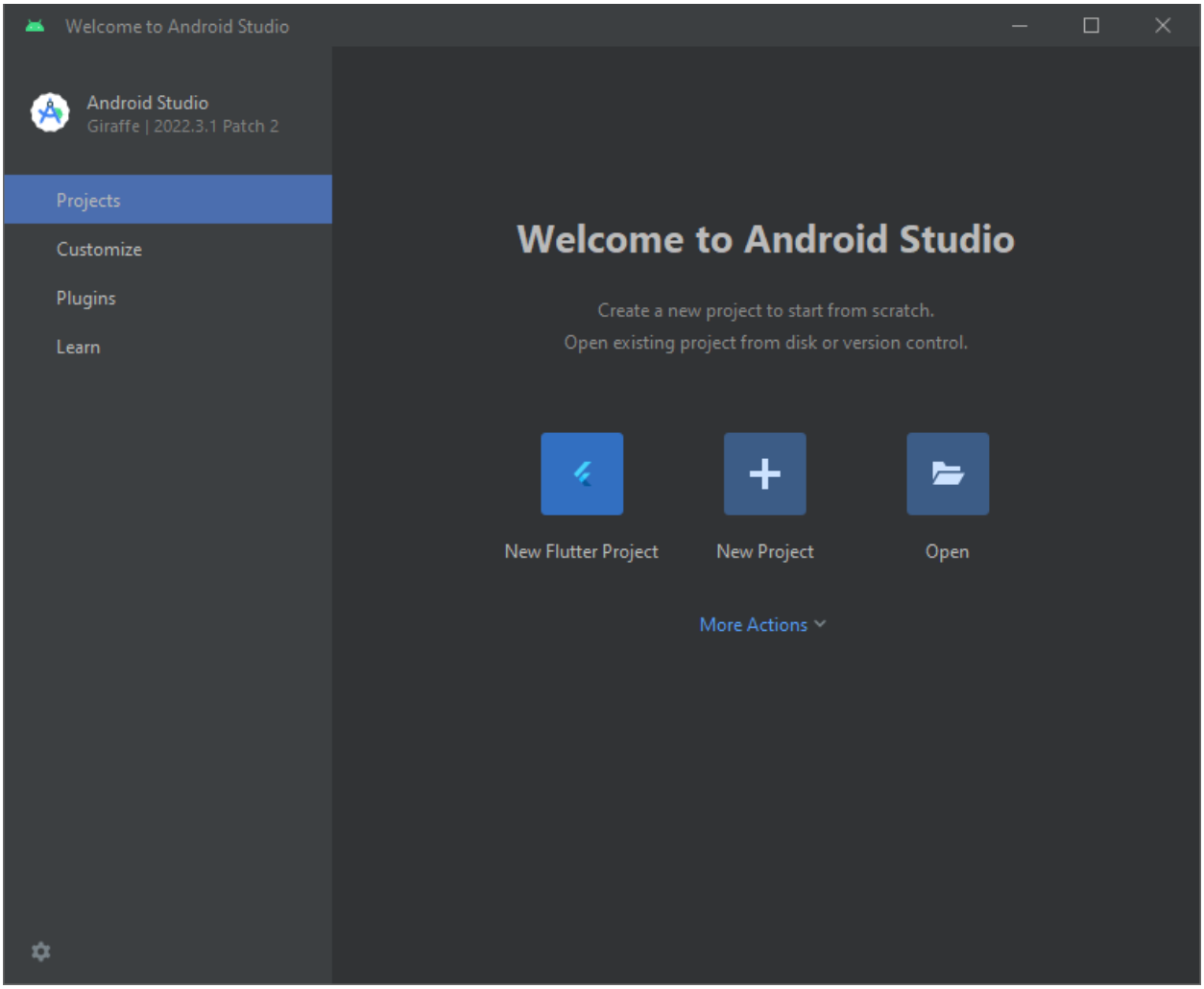
U ovoj vježbi naučiti ćete novi programski jezik pod nazivom Kotlin koji se koristi za razvoj Android aplikacija. Razmotreni su osnovni dijelovi sintakse Kotlin programskog jezika i osnove objektno orijentiranog načina programiranja sa Kotlinom.

1.1. Kotlin programski jezik

Kotlin je višeplatformski, statički tipiziran, programski jezik visoke razine opće namjene s zaključivanjem tipa. Kotlin je dizajniran za potpunu međuoperativnost s Javom, a JVM verzija Kotlinove standardne knjižnice ovisi o Java Class Library, ali zaključivanje tipa omogućuje da njegova sintaksa bude sažetija. Kotlin je također preferirani programski jezik za razvoj mobilnih aplikacija za Android operacijski sustav. Osnovna sintaksa u Kotlinu koju trebate naučiti (a koju ćemo koristiti tijekom laboratorijskih vježbi):

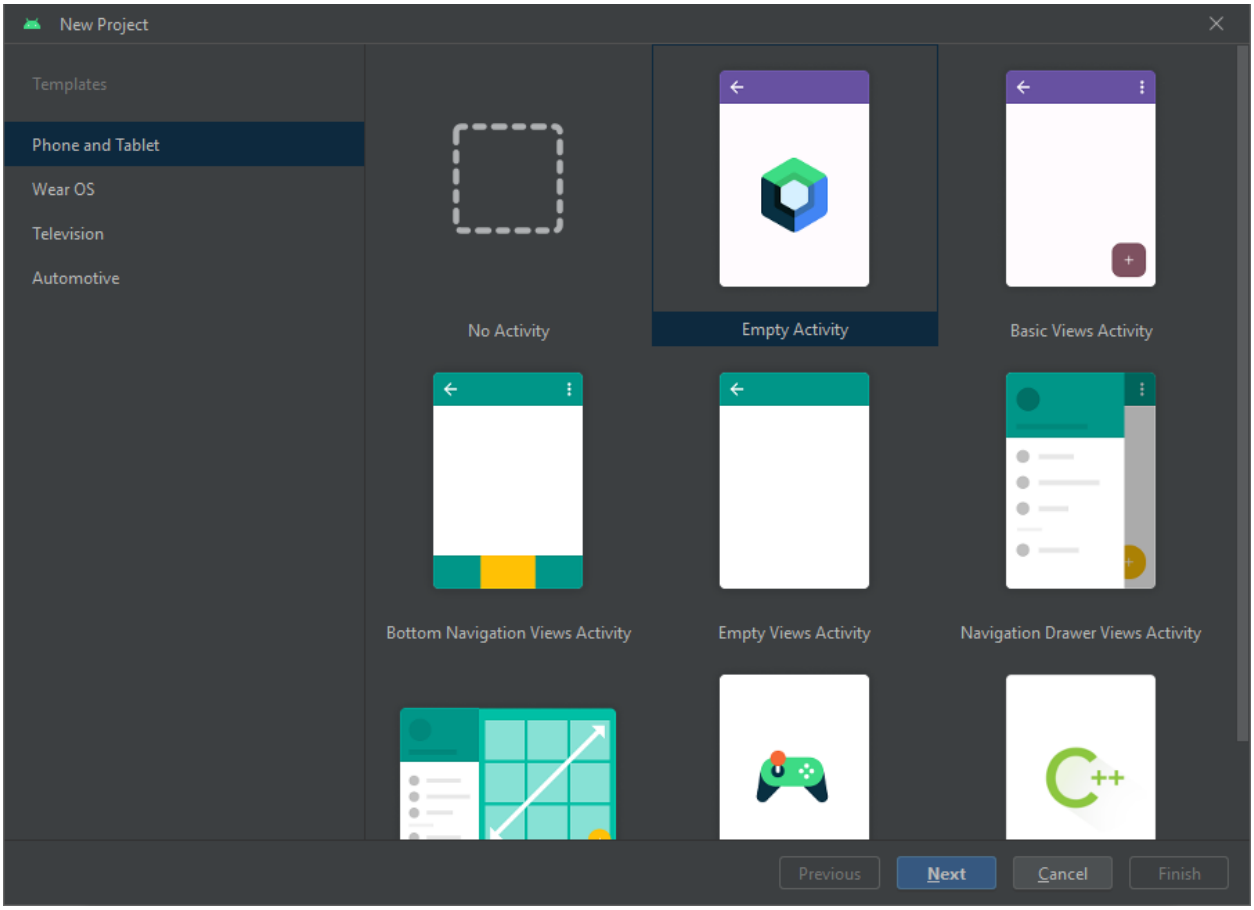
- Kao u C-u blok naredbi se definira sa vitičastim zagradama.
- Stringovi imaju dinamično formatirati koje se radi pomoću *string templates* (primjer: *println(„Hello ${myName}“))*
- Nema potrebe za točkom zarez na kraju naredbe

1.2. Stvaranje prve Android aplikacije



Slika 1.1. Prikaz glavnog izbornika unutar Android Studio programa.

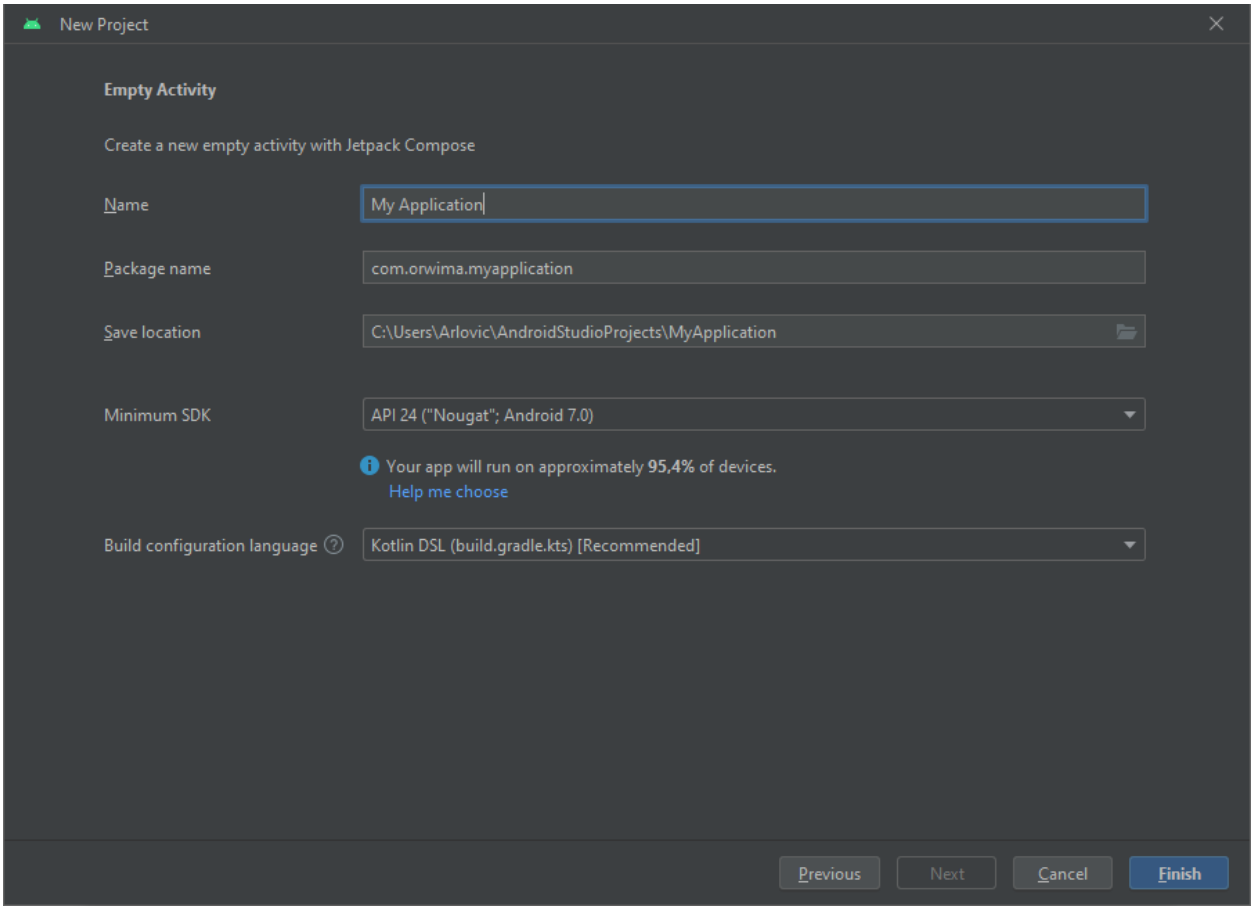
Kreiranje novog Android projekta moguće je obaviti iz glavnog izbornika kod pokretanja Android studija, kako je prikazano slikom 1.1. Klikom na *New Project* gumb otvara se novi izbornik nalik onome na slici 1.2., a koji omogućuje odabir početnog *Activitya*.



Slika 1.2. Prikaz izbornika za odabir početnog Activitya.

Ponuđeni su različiti predlošci, čak i mogućnost izostanka Activitya, a za potrebe ovih vježbi svi će novi projekti započinjati dodavanjem praznog Activitya (*Empty Activity*). Klikom na gumb *Next* korisniku se prikazuje zaslon nalik onome prikazanom na slici 1.3., koji omogućuje unos detalja vezanih uz sam projekt.

Prvenstveno se ovdje radi o nazivu aplikacije, nazivu paketa, lokaciji spremanja aplikacije, programskom jeziku u kojem se programira aplikacija i minimalnoj verziji Androida koju aplikacija podržava. Naziv aplikacije ne mora biti jedinstven u odnosu na druge aplikacije, ali naziv paketa aplikacije **MORA**. Android koristi standardnu konvenciju naziva paketa (*Package name*) još kada se koristio programski jezik Java za pravljenje Android aplikacija.

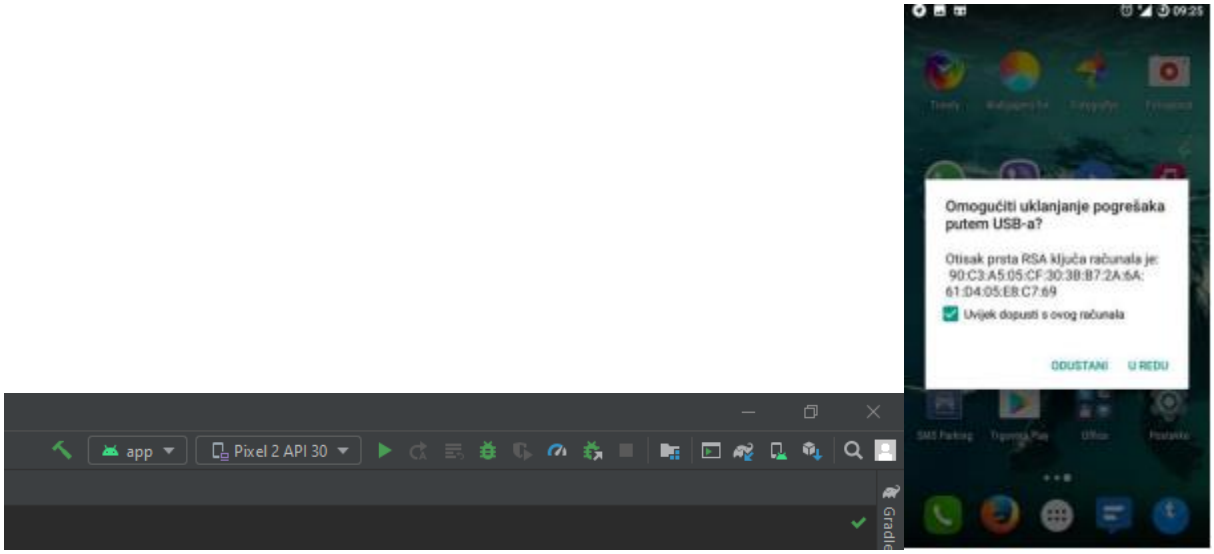


Slika 1.3. Prikaz izbornika za unos detalja o projektu.

Paket mora biti jedinstven kako bi bilo moguće razlikovati projekte, odnosno aplikacije. Ime paketa određuje se stoga upravo kao obrnuta domena, s ciljem olakšavanja održavanja jedinstvenosti. Ukoliko ne posjedujete domenu, za učenje možete unijeti bilo koji naziv paketa. Za potrebe ovih vježbi koristite sljedeću konvenciju: **hr.ferit.imeprezime.naziv_projekta**. Za razvojni jezik odaberite **Kotlin**, za minimalnu podržanu verziju Androida odaberite **Android 7.0 (Nougat)**. Nakon klika gumba *Finish* IDE generira sve potrebne klase i zapise. Nakon kraćeg čekanja se otvaraju dvije datoteke i prikazuje čitav projekt.

2.1.1. Pokretanje Android projekta

Da bi se Android projekt pokrenuo, potrebno je imati spojen i uključen uređaj. Ovdje može biti riječ ili o stvarnom ili o virtualnom uređaju. Svi spojeni uređaji dostupni preko ADB-a vidljivi su unutar Android monitora na vrhu sučelja razvojnog okruženja, kako je prikazano slikom 1.4.



Slika 1.4. Prikaz odabira uređaja.

Virtualni uređaji radit će bez ikakvih problema, dok je za korištenje fizičkih uređaja potrebno dati dozvolu na samom uređaju. Klikom na gumb u alatnoj traci ili pritiskom na kombinaciju tipku Shift i F10 pokreće se aplikacija na odabranom uređaju.

1.3. Logcat

Logcat prozor u Android Studiju pomaže vam u otklanjanju pogrešaka vaše aplikacije prikazivanjem zapisa (engl. *logs*) s vašeg uređaja u stvarnom vremenu (npr. poruke poslane iz aplikacije putem *println()* funkcije ili funkcija iz *Log* klase, poruka usluga koje rade na Androidu ili sistemskih poruka). Prilikom iznimki (engl. *exceptions*) ili rušenja aplikacije (engl. *crash*), Logcat će prikazati poruku uz cijeli stack trace i linije koda koje su uzrokovale tu iznimku ili rušenje aplikacije.

Slika 2.4. Prikaz Logcat prozora unutar Android Studija.

Svaki zapis (engl. *log*) sadrži: datum, vremensku oznaku, ID procesa i niti, oznaku, naziv paketa, prioritet i poruku. Različite oznake imaju jedinstvene boje koje pomažu pri identificiranju vrste zapisa. Svaki unos u zapisnik ima prioritet FATAL, ERROR, WARNING, INFO, DEBUG ili VERBOSE. Logcat možete pretraživati s CTRL+F.

2. RAD NA VJEŽBI

2.2. Varijable

Svi programi moraju moći pohranjivati podatke, a varijable vam u tome pomažu. U Kotlinu varijable možete deklarirati sa:

- *val* – vrijednost varijable se **ne može** mijenjati tijekom izvođenja programa
- *var* – vrijednost varijable se **može** mijenjati tijekom izvođenja programa

Za dodjeljivanje vrijednosti varijabli koristite = operator.

```
val popcorn = 5
val hotdog = 7
var customers = 10

customers = 8
println(customers)
// Izlaz programa: 8
```

Programski kod 1. Prikaz deklaracije i definicije varijabli u Kotlinu.

Svaka varijabla i struktura podataka u Kotlinu ima tip podataka. Tipovi podataka važni su jer govore prevoditelju što smijete učiniti s tom varijablom ili strukturom podataka. Drugim riječima, koje sve funkcije i svojstva posjeduje varijabla ili struktura podataka. Tip varijable definira se s dvotočkom **nakon** naziva varijable. Podržani su svi osnovni tipovi podataka (Byte, Short, Int, Long, Float, Double, Boolean, Char, String, itd.).

```
val popcorn: Int = 5
val hotdog: Float = 7.0f
var customers = 10

customers = 8
println(customers)
// Izlaz programa: 8
```

Programski kod 2. Prikaz deklaracije i definicije varijabli uz definiranje tip podatka u Kotlinu.

Prilikom programiranja korisno je moći grupirati podatke u strukture za kasniju obradu. Kotlin nudi zbirke upravo za tu svrhu.

Tip kolekcije	Opis
Liste (*List<TipPodatka>*)	Poredana zbirka predmeta
Setovi (*Set<TipPodatka>*)	Jedinstvena ne poredana zbirka predmeta
Mape (*Map<TipPodatka>*)	Skupov ključ-vrijednost parova gdje su ključevi jedinstveni i preslikavaju se na samo jednu vrijednost u zbirci

2.3. Funkcije

Funkcije unutar Kotlina deklariraju se s ključnom riječi *fun*. Parametri funkcije pišu se unutar zagrada nakon naziva funkcije.

```
fun GreetingPreview(): Unit {
    ORWIMALV1Theme {
        Greeting("Android")
    }
}
```

Programski kod 3. Prikaz deklaracije i definicije funkcija u Kotlinu.

Prilikom definiranja argumenata funkcije piše se prvo ime argumenta, a zatim njegov tip. Argumenti funkcije mogu imati podrazumijevane vrijednosti. Ove vrijednosti koriste se kada se funkcija poziva bez navedenog argumenta. Povratni tip podatka piše se nakon definiranja argumenta, a razdvojen je s dvotočkom od zagrada. Pri pozivu funkcije moguće je navesti ime svakog argumenta koji joj se predaje. Koristeći ovaj način, argumenti se ne moraju navesti redoslijedom kojim su definirani. Tijelo funkcije počinje i završava s vitičastim zagradama ({}). Ključna riječ return koristi se za izlazak iz funkcije ili vraćanje neke vrijednosti iz funkcije. Ako funkcija ne vraća ništa bitno ili je povratan tip Unit (što je kao void u C-u) onda se ne mora pisati return.

2.4. Kontrola toka programa

Kontrola toka odnosno granjanje je sastavni dio svakog programa. Najčešći oblik kontrole toka je *if-else* granjanje. *If-else* može se koristiti standardno kao tvrdnja, ili kao izraz koji vraća vrijednost, ako se koristi kao izraz, mora postojati else grana. Grane mogu sadržavati blokove koda - u ovom slučaju zadnji izraz predstavlja vrijednost tog bloka.

```
// Traditional usage
val a: Int = 5
val b: Int = 2
var max = a
if (a < b) max = b

// With else
var max: Int
if (a > b) {
    max = a
} else {
    max = b
}
// As an expression
val max = if (a > b) a else b
```

Programski kod 4. Prikaz korištenja if-else grananja.

When izraz se koristi kao ekvivalent switch-case. Kao i *if-else* može se koristiti kao tvrdnja ili kao izraz koji vraća vrijednost. *When* uspoređuje zadani parametar sa svim granama sve dok uvjet nije zadovoljen, ako se koristi kao izraz, vrijede ista pravila kao za *if-else*. Također može imati else granu koja se evaluira ako niti jedan uvjet prije nije zadovoljen.

```
when(age) {
    in 0..18 -> println("Maloljetan")
    else -> println("Punoljetan")
}

when {
    number.isOdd() -> println("Number is odd")
    else -> println("Number is even")
}
```

Programski kod 5. Prikaz korištenja when grananja.

2.5. Petlje

Kotlin koristi standardne *for*, *while* i *do-while* petlje za iteraciju kroz razne strukture podataka. Te petlje bi trebali već znati iz prošlih godina fakulteta.

```
for(item in collection) println(item)

for(i in 1..10) {
    print(i)
}

for(i in 1 until 10) {
    print(i)
}

for(i in 10 downTo 0 step 2) {
    println(i)
}

while (x > 0) {
    x--
}

do {
    val y = retrieveData()
} while (y != null)
```

Programski kod 6. Prikaz korištenja petlji.

2.6. Null vrijednost u Kotlinu

Null vrijednost opisuje vrijednost koja ne postoji, ako se pokuša pristupiti vrijednosti određene varijable u memoriji izlaz programa će biti NullPointerException. Kotlin olakšava rad s null vrijednostima. Varijabla može sadržavati null vrijednost isključivo ako eksplicitno kažemo korištenjem *?* znaka nakon tipa podatka.

```
val name: String = "Matej"      // non-nullable
val lastName: String? = null    // nullable
```

Programski kod 7. Prikaz definiranja nullabilne varijable.

Također možete lako provjeriti sadrži li varijabla null vrijednost sa *?.* operatorom. On će prvo provjeriti jeli vrijednost varijable null, ako nije onda će nastaviti s izvođenjem programa, a ako je onda će automatski vratiti null vrijednost. Bez korištenja *?.* operatora program bi se srušio sa NullPointerException pogreškom.

```
val lastName: String? = "Arlovic"     // nullable
println(lastName?.length)             // Output: 7

val lastName: String? = null          // nullable
println(lastName?.length)             // Output: null
```

Programski kod 8. Prikaz ispisa broja znakova u varijabli ako ona nije null.

Ako želite garantirati kompajleru da vrijednost varijable neće biti null onda koristite *!!.* operator. Ako vrijednost varijable tokom izvođenja programa ipak bude null, program će se srušiti sa NullPointerException progreškom. Kako bi se spriječili iznenadni padovi programa ili aplikacije koristi se elvis operator. Elvis operator (*?:*) provjerava je li uvjet s lijeve strane izraza jednak null, ako je, izvršava se desna strana, ako nije desna strana se ignorira i izvršava se lijeva strana.

```
val lastName: String? = null          // nullable
println(lastName?.length ?: 0)        // Output: 0
```

Programski kod 9. Prikaz korištenja elvis operatora.

2.7. Klase

Kotlin podržava objektno orijentirano programiranje s klasama i objektima. Klase vam omogućuju da deklarirate skup karakteristika za objekt. Kada kreirate objekte iz klase, možete uštedjeti vrijeme i trud jer ne morate svaki put deklarirati te karakteristike. Za deklaraciju klase upotrijebite ključnu riječ *class*. Deklaracija klase sastoji se od:

- imena klase
- parametara ukoliko su potrebni
- konstruktora ukoliko je potreban
- tijela klase koje se piše unutar vitičastih zagrada
- tijelo klase nije obavezno i može se izostaviti

Već smo vidjeli da klase ne moraju definirati konstruktor eksplicitno. U tom slučaju kompajler će sam napraviti podrazumijevani konstruktor koji neće primati nikakve argumente i bit će vidljiv svima. Osim toga, može postojati i više konstruktora za istu klasu. Svaka klasa ima jedan primarni konstruktor, a svi ostali (ako postoje), nazivaju se sekundarnim konstruktorima.

Svaki sekundarni konstruktor mora delegirati svoj poziv primarnom konstruktoru - koristeći this ključnu riječ. Inicijalizacijski blokovi pišu se u vitičastim zagradama ispred kojih se piše ključna riječ *init*. Init blokovi se izvršavaju neposredno nakon instanciranja objekta neke klase i u njima je moguće koristiti parametre primarnog konstruktora.

```
class Contact(name: String, lastName: String) {
    private val fullName: String
    init {
        fullName = "$name $lastName"
    }
}

class County(val name: String, val areaSqKm: Int) {
    constructor(name: String) : this(name, 0) {
        println("Secondary constructor")
    }
}
```

Karakteristike objekata neke klase mogu se deklarirati u svojstvima (engl. *properties*). Svojstva se mogu deklarirati unutar zagrada nakon naziva klase:

```
class Contact(val id: Int, var email: String
```

Ili unutar tijela klase:

```
class Contact(val id: Int, var email: String) {
    val category: String = ""
}
```

Programski kod 10. Prikaz deklaracije i definicije klasa u Kotlinu.

Svojstva imaju podrazumijevane *gettere* i *settere* ako ih sami ne definiramo. Također ih možete deklarirati i bez val i var ključnih riječi, ali tada svojstvo ne možete pozvati izvan same klase (potrebno je napisati vlastite gettere i settere). Instanca klase može se napraviti unutar programskog koda korištenjem konstruktora. Kotlin automatski stvara konstruktor s parametrima deklariranim u zaglavlju klase.

```
class Contact(val id: Int, var email: String)

fun main() {
    val contact = Contact(1, "mary@gmail.com")
}
```

Programski kod 11. Prikaz stvaranja objekta klase u Kotlinu.

Kotlin ima klase podataka (eng. *data class*) koje su posebno korisne za pohranu podataka. Klase podataka imaju istu funkcionalnost kao i obične klase, ali automatski dolaze s dodatnim funkcijama. Ove funkcije omogućuju vam jednostavan ispis instance u čitljiv izlaz, usporedbu instanci klase, kopiranje instanci i više. Kako su te funkcije automatski dostupne, ne morate trošiti vrijeme na pisanje istog šablonskog koda za svaku svoju klasu.

```
data class User(val name: String, val id: Int)
```

Programski kod 12. Prikaz definiranja klasa podataka objekta klase u Kotlinu.

Postoje četiri modifikatora vidljivosti u Kotlinu:

- *public* - vidljivo svugdje
- *private* - vidljivo unutar datoteke ili klase u kojoj je definirano
- *protected* - vidljivo unutar klase i u podklasama
- *internal* - vidljivo unutar istog modula

Podrazumijevana vidljivost je *public*.

2.8. Companion objekt

Klase u Kotlinu mogu definirati tzv. companion object blokove. Svojstvima i metodama definiranim unutar companion object bloka moguće je pristupati koristeći samo ime klase.

```
class CourseFragment {
    companion object {
        fun newInstance() = CourseFragment()
    }
}
val fragment = CourseFragment.newInstance()
```

Programski kod 13. Prikaz deklariranja i korištenja companion objekta.

3. ZADACI ZA SAMOSTALAN RAD

1. Potrebno je napraviti dvije data klase *Recipe* i *Ingredient*. Recipe klasa mora sadržavati: *image* (koji će imati *@DrawableRes* anotaciju prije ključne riječi val), *title*, *category*, *cookingTime*, *energy*, *rating*, *description*, *reviews*, *ingredients* (lista Ingredient data klase). Ingredient klasa mora sadržavati: *image* (koji će imati *@DrawableRes* anotaciju prije ključne riječi val), *title*, *subtitle*.

2. Potrebno je napraviti klasu Robot unutar koje ćete deklrarirati i definirati 2 varijable (*x* i *y*) i 5 funkcija: *goRight(steps)*, *goLeft(steps)*, *goDown(steps)*, *goUp(steps)* i *getLocation()*. Funkcije *goRight()* i *goLeft()* mijenjaju vrijednost *x* varijable, a funkcije *goUp()* i *goDown()* mijenjaju vrijednost *y* varijable. Funkcija getLocation() treba ispisati trenutnu vrijednost *x* i *y* varijabli u obliku *(x, y)*. Zatim u glavnoj Kotlin datoteci implementirajte klasu tako da na kraju izvođenja programa ispis trenutne lokacije bude **(5, 9)**.