Android akademija 2024

GDG AMA FERIT FERIT

Što je Android akademija?

Ideja stvorena 2017. (student!)

Od zajednice za zajednicu

2 generacije, ’17 (FFOS), ’18 (FERIT)

> 100 kandidata

> 40 polaznika

Nekadašnji polaznici, današnji predavači

Tko smo mi?

Bruno Zorić | Luka Kordić | Leo Svjetličić |

David Takač | Terezija Umiljanović |

Filip Babić | Marin Tolić | Martin Zagoršćak |

Goran Luketić

Google developers group Osijek | AMA-FERIT

Tko ste vi?

Može potpis?

https://forms.gle/9SxQQywXPm2HbYwy9

Kuda vodi ovaj QR?

Zašto ste vi tu?

Želite naučiti nešto o razvoju za Android

Želite upoznati druge ljude

Želite pomoći drugima

A kako?

Pridružite se Discordu

Poslušajte predavanja

Budite redoviti

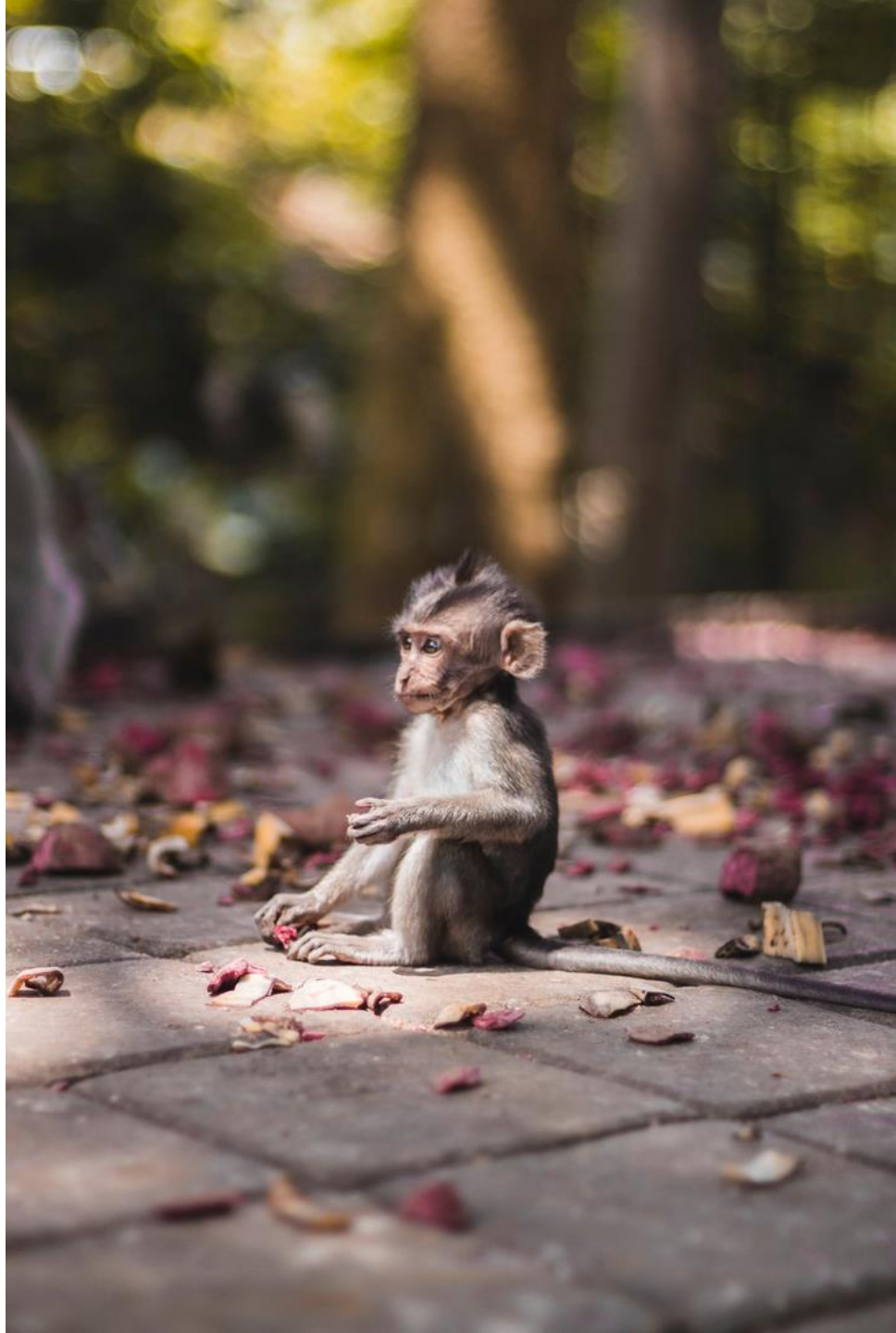
Rješavajte primjere

Rješavajte zadaće

Može prezentacija? A kod? A kava?

https://github.com/gdg-osijek/AndroidDevAcademy2024

alumni.ferit.hr



Uvod u Kotlin i OOP

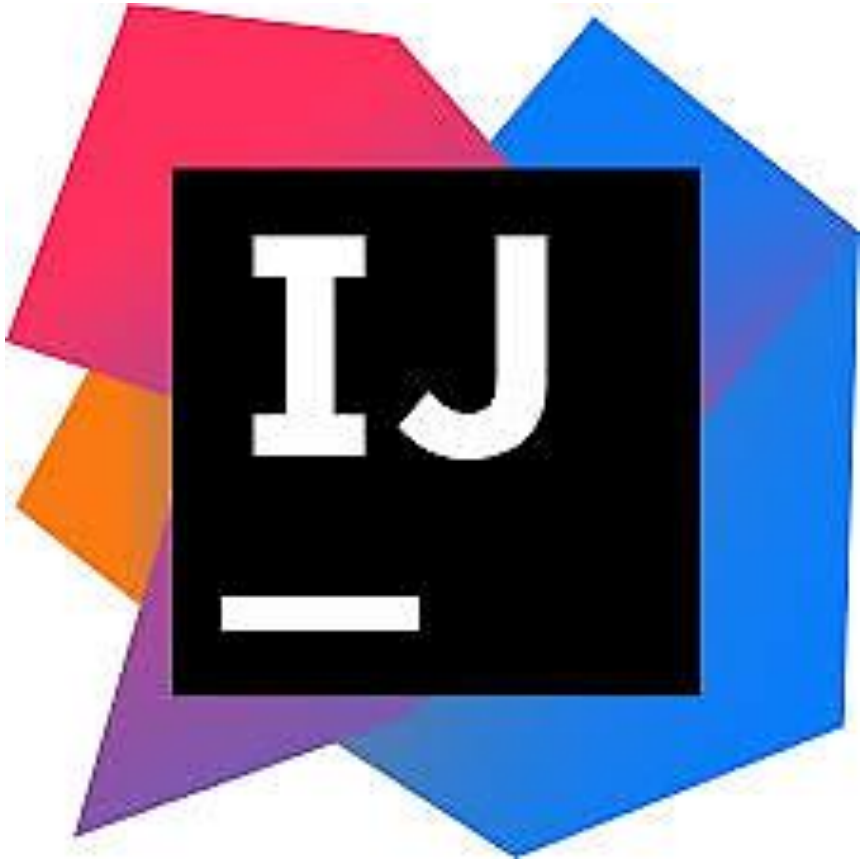
Što nam treba?

IntelliJ Idea

https://www.jetbrains.com/idea/

Ako danas nemate

https://play.kotlinlang.org/



Imate pitanje?

Sve što ste uvijek htjeli znati,

a niste se usudili pitati

Što je Kotlin?

Moderan objektno orijentiran programski jezik

Razvila ga je tvrtka JetBrains

Rabi se za razvoj raznih vrsta aplikacija, u našem slučaju posebno su važne Android aplikacije

Kompatibilan s Javom

Vrijednosti i varijable

Dvije vrste varijabli

Varijable koje je moguće samo čitati označavaju se ključnom riječju val i vrijednost im može biti pridružena samo jednom, a nazivaju se vrijednostima.

Klasične varijable kojima je moguće mijenjati vrijednost po želji označavaju se ključnom riječju var, a nazivat će se varijablama.

Sve vrijednosti i varijable moraju biti inicijalizirane prije korištenja.

Vrijednosti i varijable

Code

```
fun runExample1() {
    val name = "Jon Snow"
    var amountOfKnowledge = 0
    println("Before Android academy.")
    println("$name knows: $amountOfKnowledge.")

    amountOfKnowledge = 1
    println("After Android academy.")
    println("$name knows: $amountOfKnowledge.")
}
```

Output

```
Before Android academy.
Jon Snow knows: 0.
After Android academy.
Jon Snow knows: 1.
```

Vrijednosti i varijable

Moguće je napraviti deklaraciju uz naknadnu dodjelu vrijednosti

Ako nisu inicijalizirane, prevoditelj će javiti grešku ukoliko se koriste

Preferirano je korištenje vrijednosti tako da ih treba rabiti gdje god je to moguće.

Tip nije nužno eksplicitno navoditi već je podržano automatsko određivanje tipa.

Ukoliko se želi eksplicitno navesti tip, onda se to radi nakon imena varijable navođenjem znaka : i odgovarajućeg tipa.

Vrijednosti i varijable

Code

```
fun runExample2() {
    var spies: Int
    val direwolves: Int
    spies = 10
    direwolves = 6
    // Each direwolves eats a spy
    println("There are $spies spies and $direwolves direwolves.")
    spies = spies - direwolves
    // direwolves = direwolves - 1 // Whoops! Can't kill a direwolf in Kotlin.
    println("There are $spies spies and $direwolves direwolves.")
}
```

Output

```
There are 10 spies and 6 direwolves.
There are 4 spies and 6 direwolves.
```

Vrijednosti i varijable

Saznajte više:

https://typealias.com/start/kotlin-variables-expressions-types/

Zadatak

Napravite varijable za godinu vašeg rođenja i trenutnu godinu. Izračunajte koliko godina imate. Izračunajte koliko ćete godina imati 2048.

Zadatak

Napravite tri vrijednosti, za Vaše i imena dvaju osoba pokraj Vas na akademiji. Ispišite poruke u kojoj napominjete tko sjedi lijevo, a tko desno od koga.

Grananja (if-else)

Uvjetno izvođenje koda, stvaranje „grana” izvođenja

Najjednostavniji oblik je naredba if

Ako je uvjet zadovoljen, izvršava se blok naredbi

Moguće je imati više grana uz korištenje naredbe if-else

Izvršava se samo jedna grana, odnosno jedan blok naredbi, ostale se ignoriraju

Grananja (if-else)

Code

```
fun runExample3() {
    var winterfellPopulation = 1349
    println("Welcome to Winterfell! Population: $winterfellPopulation")
    println("How many white walkers can you see?")
    val whiteWalkersVisible = readln().toInt()
    if(whiteWalkersVisible > winterfellPopulation)
        println("Winterfell is no more.")
    else
        println("For the king in the north!")
}
```

Output

```
Welcome to Winterfell! Population: 1349
How many white walkers can you see?
100
For the king in the north!
```

Izraz if-else

Zanimljiva stvar je da je u Kotlinu if-else u stvari izraz

To znači da vraća vrijednost koja može biti dodijeljena ili iskorištena na drugi način

U slučaju da se koristi na ovaj način, mora postojati else

Izraz if-else

Code

```
fun runExample4() {
    println("Being a businessman. Guide by Littlefinger.")
    println("How to organize a wedding. Step 1 - Get money.")
    val lannisterOffer = 100233
    val tyrellOffer = 123445
    val littleFingerPurseTotal = if(lannisterOffer > tyrellOffer) lannisterOffer else tyrellOffer
    println("Offers are Lannister: $lannisterOffer and Tyrell: $tyrellOffer. Taken $littleFingerPurseTotal.")
}
```

Output

```
Being a businessman. Guide by Littlefinger.
How to organize a wedding. Step 1 - Get money.
Offers are Lannister: 100233 and Tyrell: 123445. Taken 123445.
```

When

Uvjetni izraz kod kojeg se, u ovisnosti o zadanoj vrijednosti, izvodi određen blok naredbi

Sekvencijalno se provjeravaju uvjeti svih grana sve dok jedan uvjet nije zadovoljen, ostale se tada ignoriraju

Moguće definirati podrazumijevanu granu

Nalik naredbi switch

Ako se koristi kao izraz, tada vrijednost prvog zadovoljenog uvjeta postaje vrijednost izraza

Naredba / izraz when

Code

```
fun runExample5() {
    println("What is your name?")
    val name = readln()
    println("From which region do you come?")
    val region = readln().lowercase(Locale.getDefault())
    val surname = when (region) {
        "north" -> "Snow"
        "vale" -> "Stone"
        "dorne" -> "Sand"
        "riverlands" -> "Rivers"
        else -> "Commonfolk"
    }
    println("$name $surname")
}
```

Output

```
What is your name?
Bruno
From which region do you come?
North
Bruno Snow
```

Grananje

Saznajte više:

https://www.baeldung.com/kotlin/if-else-expression

https://www.baeldung.com/kotlin/when

https://kotlinlang.org/docs/control-flow.html#for-loops

https://superkotlin.com/kotlin-when-statement/

Zadatak

Napišite kod koji od korisnika zahtjeva unos 3 broja, a zatim nađite i ispišite najveći među njima.

Zadatak

Napišite kod koji od korisnika zahtjeva unos znaka, a zatim korištenjem naredbe when određuje i na ekran ispisuje je li riječ o samoglasniku.

Petlje

Petlje omogućuju opetovano izvršavanje jedne ili više naredbi

Izvršavanje se ponavlja do ispunjenja uvjeta

U Kotlinu na raspolaganju imamo

- repeat
- for (foreach)
- while
- do…while

repeat

Najjednostavniji oblik petlje

Ponavljanje radnje *n* puta

Code

```
fun runExample6(){
    repeat(3) {
        println("I will not waste loops.")
    }
}
```

Output

```
I will not waste loops.
I will not waste loops.
I will not waste loops.
```

for

Petlja for može se rabiti za iteriranje bilo čega što pruža iterator (?)

Opći oblik:

```
for (varijabla in iterabilan_objekt) {
     tijelo – blok naredbi
}
```

Neki primjeri iterabilnih objektata su rasponi, polja i drugi oblici kolekcija poput listi, *hash*-mapa i objekata bilo koje klase koja implementira oblikovni obrazac iterator, odnosno nasljeđuje sučelje *Iterable*

for

Code

```
fun runExample7() {
    for (i in (1..5)) {
        println("$i is ${if (i % 2 == 0) "even" else "odd"}")
    }
    for (i in (10 downTo 0 step 2)) {
        println("$i is ${if (i % 2 == 0) "even" else "odd"}")
    }
}
```

Output

```
1 is odd
2 is even
3 is odd
4 is even
5 is odd
10 is even
8 is even
6 is even
4 is even
2 is even
0 is even
```

while

Petlja while omogućuje ponavljanje sve dok je kontrolni izraz istinit

Opći oblik:

```
while (uvjetni_izraz) {
     tijelo – blok naredbi
}
```

Najprije se evaluira izraz, koji mora rezultirati s *true* da bi se tijelo petlje izvelo

Moguće je da se tijelo nikada ne izvede

while

Code

```
fun runExample8() {
    var dragonsHatched = 0
    println("How many dragons do you need to conquer Westeros?")
    var dragonsNeeded = readln().toInt()
    while (dragonsHatched < dragonsNeeded) {
        println("Hatching dragon ${dragonsHatched+1}")
        dragonsHatched++
    }
    println("Total dragons hatched = $dragonsHatched/$dragonsNeeded")
}
```

Output

```
How many dragons do you need to conquer Westeros?
3
Hatching dragon 1
Hatching dragon 2
Hatching dragon 3
Total dragons hatched = 3/3
```

do...while

Petlja do...while inačica je petlje *while* kod koje se uvjet provjerava nakon svake iteracije

Opći oblik:

```
do {
     tijelo – blok naredbi
} while (uvjetni_izraz)
```

Najprije se izvodi tijelo petlje, a zatim se provjerava kontrolni uvjet za izvođenje iduće iteracije

Tijelo se uvijek izvodi barem jednom

do...while

Code

```
fun runExample9() {
    var dragonsHatched = 0
    println("How many dragons do you need to conquer Westeros?")
    var dragonsNeeded = readln().toInt()
    do {
        println("Hatching dragon ${dragonsHatched+1}")
        dragonsHatched++
    } while (dragonsHatched < dragonsNeeded)
    println("Total dragons hatched = $dragonsHatched/$dragonsNeeded")
}
```

Output

```
How many dragons do you need to conquer Westeros?
0
Hatching dragon 1
Total dragons hatched = 1/0
```

Petlje

Saznajte više:

https://www.baeldung.com/kotlin/loops

https://kotlinlang.org/docs/control-flow.html#for-loops

https://kt.academy/article/kfde-for

Zadatak

Napišite program koji uz pomoć for petlje računa i na ekran ispisuje sumu prvih *n* prirodnih brojeva, gdje se n zadaje s tipkovnice.

Zadatak

Napišite program koji od korisnika traži unos broja unutar granica zadanih također korisničkim unosom. Unos se ponavlja sve dok se ne unese broj unutar željenih granica. Kada je unesen broj, potrebno je pronaći njegovu najveću znamenku.

Funkcije

Samostalne, modularne jedinice koda

Dobro definirano sučelje, predstavljaju jedinicu posla koja se može iznova iskoristiti

Opći oblik:

```
fun ime_funkcije (formalna_lista_parametara): povratni_tip {
      tijelo funkcije – blok naredbi
}
```

Funkcije

Code

```
fun runExample10(){
    displayQuote()
    displayGreeting("Bruno")
    println(generateName("Luka"))
    println(generateName("Filip"))
    println(generateName("Terezija"))
    displayGreeting(generateName("Martin"))
}

fun displayQuote() {
    println("When you play the game of thrones, you win or you die.")
}

fun displayGreeting(name: String) {
    println("Welcome to the wedding, lord $name.")
}

fun generateName(name: String): String {
    if(name.length < 5) return "$name Darkeagle" " // Do not use magic numbers!
    if(name.length < 8) return "$name Ironheart"
    return "$name Sunseeker"
}
```

Funkcije

Output

```
When you play the game of thrones, you win or you die.
Welcome to the wedding, lord Bruno.
Luka Darkeagle
Filip Ironheart
Terezija Sunseeker
Welcome to the wedding, lord Martin Ironheart.
```

Moguće je rabiti podrazumijevane vrijednosti za argumente

Kotlin dopušta imenovane argumente (poput Pythona), tako da se eksplicitno prilikom poziva vidi koja vrijednost se rabi za koji argument

Funkcije

Saznajte više:

https://kotlinlang.org/docs/functions.html

https://www.programiz.com/kotlin-programming/functions

Zadatak

Napišite funkciju koja računa *n*-tu potenciju predanog joj broja. Rabiti for petlju, potencija i broj su predstavljeni parametrima funkcije. Testirajte napisanu funkciju.

Zadatak

Napišite funkciju koja za predan broj određuje je li on prost ili ne. Prost broj djeljiv je samo s brojem 1 i sa samim sobom. Broj predstavlja parametar funkcije. Koristiti neku od petlji obrađenih ranije.

Stringovi

Nizovi znakova predstavljeni su tipom podatka String

Pristup elementima moguće je preko indeksnog operatora []

Stringovi su nepromjenjivi, svaka izmjena stringa stvara novi string

Stringovi se mogu spajati, čime nastaje novi string, a za te potrebe rabe se najčešće operator + ili interpolacija stringova (kao ranije na predavanjima)

Stringovi

Code

```
fun runExample11(){
    println(calculatePoints("Dracarys"))
    println(calculatePoints("The night watch"))
}

fun calculatePoints(word: String): Int {
    val pointsPerVowel = 2
    val pointsPerSegment = 3
    val segmentLength = 3

    val segmentPoints = (word.length / segmentLength) * pointsPerSegment
    var vowelPoints = 0
    val vowels = setOf('a','e','i','o','u')
    for (letter in word) {
        if(letter in vowels) {
            vowelPoints += pointsPerVowel
        }
    }
    return vowelPoints + segmentPoints
}
```

Output

```
10
21
```

Stringovi

Saznajte više:

https://kotlinlang.org/docs/strings.html

https://www.baeldung.com/kotlin/strings-series

Zadatak

Napišite funkciju koja za predani string provjerava bi li on činio dovoljno jaku lozinku. Da bi to činio, mora sadržavati barem jedno veliko slovo, jedan broj te biti dug barem 8 znakova. Testirajte funkciju s nekoliko različitih stringova.

Zadatak

Napišite funkciju koja za predani string broji koliko samoglasnika sadrži. Testirajte funkciju s nekoliko različitih stringova.

Zadatak

Napišite funkciju koja za predani string broji koliko jedinstvenih znakova sadrži. Testirajte funkciju s nekoliko različitih stringova.

OOP

Paradigma razvoja

- Apstrakcija
- Enkapsulacija
- Nasljeđivanje
- Polimorfizam

Ideja je prilagoditi programski jezik problemu koji se rješava uvođenjem novih tipova podataka koji grupiranju karakteristike i radnje entiteta uočenih ili u domeni problema ili u domeni rješenja

Apstrakcija

Predstavljanje koncepta iz domene problema njegovim karakteristikama i očekivanim radnjama

Definiranje očekivanja

Žarulja? Lik u videoigri? Zmaj u GoT?

Abstraction: Character
Characteristics: • Name • HP
Responsibilities: • Check if alive • Take damage

Klase

Korisnički definiran tip podatka

Sastoje se od stanja (predstavljeno atributima / svojstvima) i ponašanja (predstavljeno metodama)

Klasa je koncept, ideja, nacrt prema kojem se grade objekti

Definira se korištenjem ključne riječi class nakon čega se navodi primarni konstruktor i u vitičastim zagradama definira stanje i ponašanje klase

Objekti

Stvarna realizacija klase

Predstavlja stvarni, fizički entitet, zauzima prostor (memoriju)

Nastaje procesom instanciranja klase, naziva se instancom klase

Moguće je deklarirati varijable tipa klase (reference) i zatim preko referenci pristupati objektima tražeći od njih usluge (poziv funkcije/metode na objektu)

Objekti

Code

```
class Dragon constructor(health: Int, name: String){
    var health: Int = health
    val name: String = name
}
```

Code

```
class Dragon(var health: Int, val name: String)
```

Code

```
fun runExample12(){
    val drogon = Dragon(100, "Drogon")
    println(drogon.name)
}
```

Output

```
Drogon
```

Stanje

Uključuje sva svojstva objekta kao i trenutne vrijednosti tih svojstava

Vrijednosti koje svojstva poprimaju mogu biti jednostavne (npr. količina i sl.) ili mogu biti kompleksne (npr. drugi objekti)

Uobičajeno se predstavlja atributima, Kotlin ih nema pa se rabe svojstva

Svaki objekt ima vlastito stanje, odnosno vlastite atribute

Stanje

Opći oblik:

```
var ime_svojstva: Tip = inicijalizator_svojstva

        pravo_pristupa get = implementacija

        pravo_pristupa set(value) = implementacija
```

Češće korišteno:

```
var ime_svojstva = inicijalizator_svojstva
```

Ponašanje

Ponašanje je način na koji se objekt ponaša i reagira u smislu promjene njegova stanja ili slanja poruka drugim objektima.

Uobičajeno se implementira pomoću funkcija deklariranih unutar klase (funkcijski članovi, metode).

Enkapsulacija

Proces grupiranja podataka i metoda koje na njima rade u klasu te razdvajanja javnog sučelja apstrakcije od njegove implementacije pri čemu implementacija mora biti skrivena.

Apstrakcija i enkapsulacija su komplementarne

- Apstrakcija mora prethoditi enkapsulaciji.
- Da bi apstrakcija bila moguća, implementacije mora biti enkapsulirana.

Najčešće se ostvaruje skrivanjem informacija.

- Skriva se struktura klase, kao i implementacija metoda.

Prava pristupa

Najjednostavniji način za ostvarivanje enkapsulacije

Skrivanje implementacije

Ograničavanje pristupa članovima klase

U Kotlinu 4 tipa

- public
- private
- protected
- internal

Stanje i ponašanje

Code

```
class Sword(
    val name: String,
    val damage: Int = 1,
    val durability: Int = 100
) {
    var hitsRemaining: Int = durability
        private set(value) {
            field = value
    }

    fun attack(hits: Int): Int {
        val hitsToDo = Math.min(hits, hitsRemaining)
        hitsRemaining -= hitsToDo
        return hitsToDo * damage
    }

    fun repair() { hitsRemaining = durability }

    override fun toString(): String {
        return "Sword $name, attack damage: $damage, duration: $hitsRemaining."
    }
}
```

Stanje

Code

```
fun runExample13() {
    val longclaw = Sword("Longclaw", 1)
    val dawn = Sword(name = "Dawn", damage = 3, durability = 300)
    println(longclaw)
    println(longclaw.attack(hits = 12))
    println(longclaw)
    longclaw.repair()
    println(longclaw)
}
```

Output

```
Sword Longclaw, attack damage: 1, duration: 100.
12
Sword Longclaw, attack damage: 1, duration: 88.
Sword Longclaw, attack damage: 1, duration: 100.
```

Konstruktori

Da bi se objekti mogli rabiti (upućivati im zahtjeve, odnosno na njima pozivati metode), moraju biti stvoreni

Stvaranje objekta = instanciranje klase

Poziva se posebna metoda klase koja se zove konstruktor

Konstruktor se poziva prilikom svakog kreiranja objekta

Dovođenje objekta u stanje gdje je s njime sigurno raditi

Skupljanje smeća osigurava ispravno rukovanje memorijom

Konstruktori

Kotlin poznaje primarni ctor, sekundarne ctore i blok za inicijalizaciju

Blok za inicijalizaciju (init{}) izvodi se odmah nakon primarnog konstruktora, a služi za dodavanje koda primarnom konstruktoru

Inicijalizacija u inicijalizacijskom bloku vrijedi za sve konstruktore

Uobičajeno se sekundarni konstruktori pišu kao tzv. delegirani konstruktori, odnosno posao se delegira primarnom konstruktoru pozivom *this()* i prosljeđivanjem argumenata odmah nakon formalne liste parametara

Konstruktori

Alokacija memorije

Zero-out

Konstruktor osnovnog tipa

Primarni konstruktor

Inicijalizatori svojstava i init blokovi

Sekundarni konstruktor

Konstruktori

Code

```
class Character(
    val name: String,
    val isValyrian: Boolean = false,
    var strength: Int,
    var intelligence: Int,
) {
    constructor() : this("Unknown", false, 1, 1) {}
    constructor(name: String) : this(name, false, 1, 1) {}

    val dexterity: Int = intelligence * 2 - strength // Initializer

    // Init block:
    init {
        if (isValyrian) {
            this.strength *= 2
            this.intelligence *= 2
        }
    }

    // Expression bodied method:
    override fun toString(): String = "$name ${if (isValyrian) "of Valyria" else ""} S:$strength I:$intelligence "
}
```

Konstruktori

Code

```
fun runExample14() {
    val danaerys = Character("Danaerys", true, 1, 8)
    val jorah = Character("Jorah", false, 10, 2)
    println(danaerys)
    println(jorah)
}
```

Output

```
Danaerys of Valyria S:2 I:16
Jorah  S:10 I:2
```

Klase i objekti

Saznajte više:

https://www.baeldung.com/kotlin/constructors

https://kotlinlang.org/docs/properties.html#getters-and-setters

https://kotlinlang.org/docs/classes.html

https://kt.academy/article/kfde-functions

https://www.programiz.com/kotlin-programming/constructors

Zadatak

Definirajte klasu koja predstavlja krug, čije stanje čini polumjer kruga a ponašanje uključuje izračun površine i opsega kruga. Omogućite stvaranje jediničnog kruga kao i kruga sa zadanim polumjerom. Napišite testni program za klasu krug.

Zadatak

Definirajte klasu koja predstavlja točku u 2D prostoru čije stanje čine x i y koordinata. Omogućite stvaranje točaka podrazumijevanim konstruktorom, konstruktorom koji obje koordinate postavlja na istu vrijednost kao i točke s proizvoljnim koordinatama. Definirajte metodu za translaciju koja za predan pomak u x i y smjeru pomiče točku. Definirajte metodu koja računa euklidsku udaljenost između trenutne i točke zadane parametrom metode. Napišite testni program za klasu koja predstavlja točku.

object

Često se javlja potreba da postoji samo jedna instanca neke klase

Ranije se ovo rješavalo obrascem Singleton

Kotlin to ima riješeno kao dio jezika

Korištenjem ključne riječi object kreira se jedinstveni objekt – kreira se klasa i instancira objekt u istom koraku

Objektu se pristupa direktno preko imena, ne instancira se klasa

Nema konstruktor, postoji samo jedan

object

Code

```
object TheIronThrone {
    var swordCount = 1000

    fun meltSwordOfFallenEnemy() { swordCount++ }

    fun meltSwordOfFallenEnemy(swordCount: Int) { this.swordCount += swordCount }
}

fun runExample15() {
    println("Total swords in the throne: ${TheIronThrone.swordCount}")
    println("Dragons burn down enemy")
    TheIronThrone.meltSwordOfFallenEnemy()
    println("Total swords in the throne: ${TheIronThrone.swordCount}")
}
```

Output

```
Total swords in the throne: 1000
Dragons burn down enemy
Total swords in the throne: 1001
```

Companion object

Svo do sada viđeno stanje i ponašanje bilo je dijelom nekog objekta

Takvo stanje naziva se varijablama instance, a metode metodama instance

Često se javlja potreba da se varijable i metode definiraju neovisno o instanci, u drugim jezicima to se postiže korištenjem ključne riječi static

Kotlin ima na raspolaganju takozvane companion objekte

Moguće im je pristupati eksplicitno ili implicitno

Companion object

Code

```
class Army private constructor(val soldiers: Int) {
    companion object {
        const val PRICE_PER_SOLDIER = 3

        fun hire(money: Int): Army {
            return Army(money / PRICE_PER_SOLDIER)
        }

        fun draft(population: Int): Army {
            return Army(population / 2)
        }
    }
}
```

Companion object

Code

```
fun runExample16() {
    val borrowedGold = 1000
    val unsullied = Army.hire(borrowedGold);
    println("Unsullied: ${unsullied.soldiers}.")
    val winterfellPopulation = 1045
    val nightsWatch = Army.draft(winterfellPopulation);
    println("Nights Watch: ${nightsWatch.soldiers}.")
}
```

Output

```
Unsullied: 333.
Nights Watch: 522.
```

Nullability

Svi tipovi u Kotlinu su takozvani tipovi reference (engl. *reference types*)

Ovo znači da varijabla i vrijednost nisu jedno te isto, varijabla je referenca na stvarnu vrijednost koja se nalazi negdje na hrpi

Posredni pristup objektima

Neinicijalizirana referenca ima vrijednost *null*, a pristup članovima objekta preko takve reference dovodi do NPE (*null pointer exception*)

Nullability

Code

```
fun pickSword(selection: Int): Sword? {
    return when(selection){
        1 -> Sword("Dawn", 10, 100)
        2 -> Sword("Longclaw", 5, 100)
        else -> null
    }
}

fun findSword(): Sword = Sword("Valyrian steel", 20, 1000)

fun runExample17() {
    // Non-nullable and nullable type:
    val longclaw: Sword = null  // Illegal
    val dawn: Sword? = null     // Legal
    // Using methods returning null
    val nullable_sword: Sword? = pickSword(1)
    nullable_sword.repair() // Cannot be used on a nullable type reference
    nullable_sword?.repair() // Safe call operator can be used
    val non_nullable_sword: Sword = pickSword(2)  // Illegal, can't combine
    // How to combine?
    val sword = pickSword(2) ?: findSword()
    sword.repair()
}
```

Kompozicija

Svaka klasa treba imati samo jednu odgovornost

Složena ponašanja ostvaruju se suradnjom objekata više klasa

Klase kao svojstva mogu imati objekte drugih klasa

Kompozicija opisuje odnos *has-a*

Automobil ima motor, hotel ima sobe, Android akademija ima predavače

Kompozicija

Code

```
un runExample18() {
    val stick = Sword("Stick", 1,3)
    println("Swords single attack: ${stick.attack(1)}")
    val syrio = Swordsman(10, true, stick)
    println("Swordsman single attack: ${syrio.attack()}")
}

class Swordsman(
    val strength: Int,
    val isBlessed: Boolean,
    val sword: Sword
) {
    private val blessedMultiplier = 2
    fun attack(): Int {
        val baseDamage = sword.attack(1) + strength
        if(isBlessed)
            return blessedMultiplier * baseDamage
        return baseDamage
    }
}
```

Output

```
Swords single attack: 1
Swordsman single attack: 22
```

Predavanje 1

Zadatak

Napišite program koji omogućuje igranje igre Jamb (engl. Yahtzee). Razdvojite funkcionalnost u klase. Trebaju postojati klasa koja predstavlja kockicu, omogućuje bacanje kockice i provjeru trenutnog stanja, klasa koja predstavlja ruku sa 6 kockica, omogućije bacanje svih kockica, zaključavanje 0-6 kockica, bacanje samo otključanih kockica, provjeru rezultata bacanja. Podržati barem 3 provjere za jamb (npr. jamb, poker, skala). Napišite funkciju za provjeru napisane funkcionalnosti.

Zadatak

Napišite program za igranje kartaške igre *War*. Kreirajte klasu koja predstavlja kartu. Za boje i vrijednosti možete rabiti takozvane *enum* klase. Kreirajte klasu koja predstavlja špil i omogućuje miješanje i dijeljenje po jedne karte. S obzirom da postoje 52 karte, možete rabiti polje (engl. *array – pogledati dokumentaciju*). Svaki igrač u svakoj rundi dobije po 1 kartu. Rundu dobiva igrač s jačom kartom (veći broj, as je najjači). Napišite funkciju koja omogućuje igranje ove igre za dva igrača.

Napredni Kotlin

Luka Kordić, 6.11.2024.

Programming is the art of telling another human being what one wants the computer to do.

— Donald Ervin Knuth