SVEUČILIŠTE U SPLITU

**Sveučilišni odjel za stručne studije**

# OBJEKTNO PROGRAMIRANJE

## Case study: Non-mainstream programski jezici

## Scala

Sanja Berić

Split, siječanj 2025.

# Sadržaj

# 1. UVOD

Scala je programski jezik opće namjene koji podržava objektno orijentirano programiranje i funkcionalno programiranje. Izvorni kod Scale može se prevesti u Java bajt kod te se izvršavati na Java virtualnom stroju (JVM), čime omogućava visoku razinu interoperabilnosti s Javom. Ova interoperabilnost znači da se biblioteke napisane na Javi mogu jednostavno koristiti u Scali, i obrnuto, čime se značajno proširuje funkcionalnost oba jezika.

Osim JVM-a, Scala se može kompajlirati u JavaScript, omogućavajući pokretanje aplikacija u preglednicima, a odnedavno i u izvorni izvršni kod za visoko optimizirane performanse. To je čini prikladnom za razne primjene, uključujući razvoj web aplikacija, analitiku podataka i skalabilne distribuirane sustave.

Scala se ističe svojom modernom sintaksom. Tradicionalno koristi vitičaste zagrade za označavanje blokova koda, slično jezicima poput C-a ili Jave. Međutim, u verziji Scala 3 uvedena je mogućnost korištenja off-side pravila (uvlačenja), što olakšava čitljivost koda i smanjuje mogućnost sintaktičkih pogrešaka. Ovo se smatra jednom od najvažnijih promjena uvedenih u Scali 3.

Scala podržava snažnu statičku tipizaciju, što omogućuje otkrivanje pogrešaka tijekom kompajliranja, čime se poboljšava pouzdanost aplikacija. Nedavno je Scala stekla popularnost u području podatkovne znanosti i strojnog učenja zahvaljujući alatima kao što su Apache Spark i ScalaNLP, koji su napisani u ovom jeziku. Zajednica Scale kontinuirano raste, a razvojni alati poput sbt-a (Scala Build Tool) i Metalsa dodatno olakšavaju rad u jeziku.

# 2. POVIJEST I RAZVOJ

Scala programski jezik osmislio je Martin Odersky, profesor metoda programiranja na École Polytechnique Fédérale de Lausanne (EPFL) u Švicarskoj i istaknuti računalni znanstvenik. Odersky je poznat i kao sukreator Java kompajlera, generičke verzije Jave te EPFL-ovog Funnel programskog jezika. Njegovo bogato iskustvo u razvoju Jave značajno je utjecalo na sličnosti između Scale i Jave, pri čemu je Scala dizajnirana da radi na Java Virtual Machineu i pruža visoku razinu interoperabilnosti.

Rad na Scali započeo je 2001. godine, a prva verzija jezika javno je objavljena 2004. na Java platformi. Druga verzija Scale, v2.0, objavljena je 2006. i donijela je značajna poboljšanja u stabilnosti i funkcionalnosti jezika. Na konferenciji JavaOne 2012. Scala je osvojila nagradu ScriptBowl, što je dodatno učvrstilo njen status inovativnog jezika unutar zajednice programera.

Martin Odersky je izjavio da se vrlo malo značajki Scale može smatrati u potpunosti "novima". Umjesto toga, inovacija Scale leži u načinu na koji su njeni konstrukti međusobno povezani. Njegova vizija bila je stvoriti "bolji jezik" koji će nadmašiti Javu u fleksibilnosti i produktivnosti.

Scala 3, objavljena 2021., donijela je značajne promjene u jeziku, uključujući uvlačenje koda (off-side rule) kao alternativu zagradama, jednostavniji tipovni sustav i unaprijeđenu podršku za metaprogramiranje. Ove su promjene osmišljene kako bi pojednostavile učenje i korištenje jezika, zadržavajući pritom njegovu snagu i fleksibilnost. Danas se Scala široko koristi u industriji, posebno u domenama obrade velikih podataka i razvoja distribuiranih sustava.

# 3. SINTAKSA

Sintaksa Scale je moderna i fleksibilna, kombinira elemente objektno-orijentiranog i funkcionalnog programiranja s naglaskom na sažetost i jasnoću.

Ključne riječi su riječi u jeziku koje se koriste za neke unaprijed definirane radnje. Ove riječi stoga nije dopušteno koristiti kao nazive varijabli ili objekte. U slučaju može doći do pogreške tijekom kompajliranja (for, if, null, import, do, class, true, return…).

Deklaracija varijabli:

- Scala koristi ključne riječi val i var za deklaraciju varijabli.
- val označava nepromjenjive varijable (slične konstantama), dok var označava promjenjive.

```
val ime: String = "Marko"   //nepromjenjivo
var godine: Int = 25        //promjenjivo
godine = 26                 //ažuriranje vrijednosti
```

Tipovi podataka:

- Kategorizacija podataka koja kompajleru govori koju vrstu vrijednosti ima varijabla.
- Scala podržava osnovne tipove podataka poput Int, Double, Boolean, Char, String, kao i složenije strukture poput kolekcija.
- Tipovi podataka se automatski zaključuju, ali se mogu eksplicitno navesti.

```
val broj = 10                   //automatsko zaključivanje (Int)
val decimalni: Double = 10.5
```

## 3.1 Funkcije

Definiranje funkcija:

- Funkcije se definiraju pomoću ključne riječi def.
- Povratni tip (return) može biti eksplicitno naveden, ali nije obavezan. Vrijednost posljednje izvršene naredbe ili izraza obično je vrijednost funkcije.

```
def zbroj(a: Int, b: Int): Int = {
        a + b
      }
println(zbroj(5, 3))
//Ispisuje 8
```

Anonimne funkcije:

- Scala omogućava pisanje anonimnih funkcija pomoću strelice =>.

```
val mnozenje = (a: Int, b: Int) => a * b
println(mnozenje(4, 5))
//Ispisuje 20
```

## 3.2 Liste

- U Scali, liste su kolekcije elemenata istog tipa koje su nepromjenjive (immutable). To znači da se jednom kreirana lista ne može mijenjati pa svako dodavanje ili uklanjanje elemenata stvara novu listu.
- Novu listu stvaramo korištenjem ugrađenog konstrutkora List.
  Npr. val brojevi = List(1, 2, 3, 4, 5)
- Iako su liste nepromjenjive, element možemo dodati na početak koristeći operator ::
  Npr.

```
val novaLista = 1 :: brojevi
 println(novaLista)
// Ispisuje: List(1, 2, 3, 4)
```

- Primjer liste:

```
object ListaPrimjer {
    def main(args: Array[String]): Unit = {
        val voće = List("jabuka", "banana", "naranča")    // Definicija liste
        println("Elementi liste: " + voće)
        println("Prvi element: " + voće.head)    // Dohvaćanje prvog elementa
        println("Ostatak liste: " + voće.tail)    // Dohvaćanje ostatka liste
    }
}

//Ispisuje:
Elementi liste: List(jabuka, banana, naranča)
Prvi element: jabuka
Ostatak liste: List(banana, naranča)
```

## 3.3 Stringovi

- U Scali, string predstavlja sekvencu znakova ili niz znakova. Radi se o indeksiranoj strukturi podataka koja koristi linearan način pohrane u memoriji.
- Poput Jave, stringovi u Scali su nepromjenjivi. To znači da se izvorni objekt niza ne može izmijeniti, ali je moguće primjenjivati različite metode na njemu kako bi se dobio željeni rezultat.

- Primjer stringova:

```
object StringExample {
    def main(args: Array[String]): Unit = {
        val greeting = "Pozdrav, Scala!"
        val name = "Korisnik"
        val message = s"$greeting Dobrodošao, $name!"  // Spajanje stringova
        println(message)
    }
}
//Ispisuje:
Pozdrav, Scala! Dobrodošao, Korisnik!
```

# 4. OBJEKTNO-ORIJENTIRANE ZNAČAJKE

## 4.1 Klase

- Klasa je korisnički definiran prototip iz kojeg se stvaraju objekti. U osnovi, u klasi se konstruktor koristi za inicijalizaciju novih objekata, polja su varijable koje daju stanje klase i njenih objekata, a metode se koriste za implementaciju ponašanja klase i njenih objekata.
- U Scali, deklaracija klase sadrži ključnu riječ class, iza koje slijedi identifikator (ime) klase. Naziv klase treba započeti početnim slovom. Zatim slijedi ime nadređene klase, ako postoji, ispred kojeg stoji ključna riječ extends. Svojstva koja implementira klasa, ako ih ima, su odvojena zarezima i ispred njih stoji ključna riječ extends. Klasa može implementirati više od jedne osobine.
- Tijelo klase okruženo je vitičastim zagradama { }.

```
// Definicija klase
class Student(val id: Int, val ime: String) {
    // Metoda za ispis podataka o studentu
    def prikazi(): Unit = {
        println(id + " " + ime)
    }
}

// Glavni objekt s glavnom metodom
object GlavniObjekt {
    def main(args: Array[String]): Unit = {
    // Stvaranje instance klase Student
        var s = new Student(90210, "Tina")
        // Pozivanje metode prikazi
        s.prikazi()
 }
}

//Ispisuje :
90210, Tina
```

## 4.2 Nasljeđivanje

- Nasljeđivanje je jedan od ključnih temelja objektno orijentiranog programiranja (OOP).
- U Scali omogućuje jednoj klasi da preuzme svojstva i metode druge klase.
- Za ostvarivanje nasljeđivanja koristi se ključna riječ extends. Nadređena klasa je ona čije se karakteristike nasljeđuju, dok je podklasa ona koja nasljeđuje te značajke.

PRIMJER POJEDINAČNOG NASLJEĐIVANJA:

```
// Nadređena klasa
class Zaposlenik {
    var placa: Float = 12000
}

// Podklasa koja nasljeđuje klasu Zaposlenik
class Menadzer extends Zaposlenik {
    var bonus: Int = 3000
    def prikazi(): Unit = {
        println("Plaća = " + placa)
        println("Bonus = " + bonus)
    }
}

// Glavni objekt
object GlavniObjekt {
    def main(args: Array[String]): Unit = {
        new Menadzer().prikazi()
    }
}

//Ispisuje:
Plaća = 12000.0
Bonus = 3000
```

PRIMJER VIŠERAZINSKOG NASLJEĐIVANJA:

```
// Prva klasa
class X {
    var prihod1 = 1500
}

// Druga klasa koja nasljeđuje klasu X
class Y extends X {
    var prihod2 = 2500
}

// Treća klasa koja nasljeđuje klasu Y
class Z extends Y {
    def prikaziPrihode(): Unit = {
        println("Prihod1 = " + prihod1)
        println("Prihod2 = " + prihod2)
    }
}

// Glavni objekt
object GlavniObjekt {
    def main(args: Array[String]): Unit = {
        var z = new Z()
        z.prikaziPrihode()
    }
}

//Ispisuje:
Prihod1 = 1500
Prihod2 = 2500
```

## 4.3 Konstruktori

- U Scali, konstruktor se ne smatra posebnom metodom. Scala omogućuje definiranje jednog primarnog i neograničenog broja pomoćnih konstruktora.
- Ako u Scali ne definirate eksplicitno primarni konstruktor, kompajler automatski generira konstruktor koji se naziva primarni konstruktor. Sve naredbe unutar tijela klase automatski se uključuju u konstruktor.
- Pomoćni konstruktor mora uvijek pozvati primarni konstruktor, a za to se koristi ključna riječ **this**. Pozivanje drugog konstruktora mora biti smješteno kao prva naredba unutar tijela pomoćnog konstruktora.

- Primjer konstruktora:

```
// Primarni konstruktor
class Korisnik(broj: Int, imePrezime: String) {
    var godine: Int = -1   // Varijabla koja će biti inicijalizirana kroz pomoćni konstruktor
    def ispisiDetalje(): Unit = {
        println(s"ID: $broj, Ime: $imePrezime, Godine: $godine")
    }

    // Pomoćni konstruktor
    def this(broj: Int, imePrezime: String, godine: Int) {
        this(broj, imePrezime)   // Pozivanje primarnog konstruktora mora biti prva linija
        this.godine = godine
    }
}

object Aplikacija {
    def main(args: Array[String]): Unit = {
        val korisnik = new Korisnik(501, "Petar Perić", 30) // Poziva pomoćni konstruktor
        korisnik.ispisiDetalje()
    }
}

//Ispisuje:
ID: 501, Ime: Petar Perić, Godine: 30
```

# 5. NAPREDNE ZNAČAJKE SCALE

## 5.1 Iznimke

- Iznimke u Scali koriste se za rukovanje greškama, slično kao u drugim jezicima poput Jave i C++. Ključne riječi `try`, `catch` i `finally` omogućuju obradu grešaka i izvršavanje kôda nakon njih, bez obzira na ishod.

- Primjer korištenja iznimki:

```
object ExceptionExample {
    def main(args: Array[String]): Unit = {
        try {
            val result = 10 / 0
            println(result)
        } catch {
            case e: ArithmeticException => println("Greška: Dijeljenje s nulom!")
            case e: Exception => println("Nešto je pošlo po zlu: " + e.getMessage)
        } finally {
            println("Blok finally se uvijek izvršava.")
        }
    }
}
```

## 5.2 Template klase i funkcije

- Scala omogućuje korištenje template (generičkih) klasa i metoda kako bi se omogućila fleksibilnost rada s različitim tipovima podataka.

- Primjer template funkcije:

```
object GenericFunctionExample {
        def add[T](a: T, b: T)(implicit num: Numeric[T]): T = {
        num.plus(a, b)
 }

 def main(args: Array[String]): Unit = {
        println(add(3, 7))       // Za cijele brojeve
        println(add(3.5, 7.2))   // Za decimalne brojeve
       }
}
```

- Primjer template klase:

```
class Box[T](private var content: T) {
        def getContent: T = content
        def setContent(value: T): Unit = {
              content = value
       }
}

object GenericClassExample {
        def main(args: Array[String]): Unit = {
              val intBox = new Box[Int](5)
              val stringBox = new Box[String]("Hello")
              println(intBox.getContent)
              println(stringBox.getContent)
      }
}
```

## 5.3 Lambda izrazi i funktori

- Lambda izrazi su anonimne funkcije koje se koriste za inline logiku. Scala podržava izraze poput (x: Int) => x * x za definiciju lambda funkcija.

- Primjer lambda izraza i funktora:

```
object LambdaExample {
      def main(args: Array[String]): Unit = {
            val numbers = List(1, 2, 3, 4, 5)
            val squared = numbers.map(x => x * x)
            println(squared)
            // Sintaktički skraćeno
           val cubed = numbers.map(_ * _ * _)
           println(cubed)
       }
}
```

## 5.4 Napredni STL

- Scala pruža snažan sustav kolekcija koji uključuje strukture poput `List`, `Set`, `Map`, te funkcije višeg reda za rad s podacima.

- Primjer rada s kolekcijama:

```
object CollectionsExample {
     def main(args: Array[String]): Unit = {
           val numbers = List(1, 2, 3, 4, 5)
           val filtered = numbers.filter(_ % 2 == 0)
           println("Parni brojevi: " + filtered)

          val mapped = numbers.map(_ * 2)
          println("Pomnoženo s 2: " + mapped)

          val reduced = numbers.reduce(_ + _)
          println("Suma: " + reduced)
     }
}
```

# 6. ZAKLJUČAK

Ovaj dokument prikazuje sve najbitnije značajke Scala programskog jezika i njihovu primjenu. Scala je programski jezik koji dijeli mnoge sličnosti s Javom, što doprinosi njegovoj popularnosti među korisnicima. Za razliku od Jave, koja se temelji isključivo na objektno orijentiranom pristupu, Scala kombinira principe objektno orijentiranog i funkcionalnog programiranja, čime nudi dodatnu razinu fleksibilnosti. Scala se izdvaja svojom potpunom kompatibilnošću s Javom. Ova karakteristika, uz prilagodljivost i mogućnost korištenja u različitim okruženjima, čini Scalu izuzetno moćnim alatom za razvoj softvera. Značajke kao što su snažna statička tipizacija, lambda izrazi, podrška za kolekcije i višerazinsko nasljeđivanje svrstavaju Scalu među napredne jezike koji omogućuju pisanje čitljivog i pouzdanog koda. Sa zajednicom u stalnom rastu i aktivnim razvojem, Scala ostaje relevantna za moderne izazove u softverskom inženjerstvu.

# 7. LITERATURA


1) https://www.scala-lang.org/
2) https://en.wikipedia.org/wiki/Scala_(programming_language)
3) https://www.scala-js.org/
4) https://www.geeksforgeeks.org/scala-programming-language/
5) https://www.contrastsecurity.com/glossary/scala-programming-language
6) https://www.simplilearn.com/scala-programming-article