ALTERNATIVNI JEZICI NA JAVA PLATFORMI

Ivan Senji, IN2 d.o.o., +385 1 6386 873, ivan.senji@in2.hr
Mario Popović, IN2 d.o.o., +385 1 6386 868, mario.popovic@in2.hr

SAŽETAK

U ovom radu opisujemo neke popularnije alternative programskom jeziku Java koje se izvode na Java virtualnoj mašini (*engl. Java Virtual Machine, JVM*). Navodimo njihove prednosti te razloge zašto ih je dobro imati u vidu i razmisliti o korištenju alternative Javi. Detaljnije i na više primjera opisat ćemo Scalu i Groovy, kako ti jezici mogu olakšati život programeru te navesti njihove prednosti i nedostatke.

1. UVOD

Java je programski jezik kojeg je razvio James Gosling iz Sun-a i središnji je dio (nekad Sun-ove, sada Oraclove) Java platforme. Java je sintaksom slična jezicima C i C++, ali je u usporedbi s C++-om znatno jednostavniji jezik; sa jednostavnijim objektnim modelom i manjom mogućnosti pristupanja sustavima niže razine (memoriji, procesoru i slično). Java programski kod se najčešće prevodi u bytecode i izvršava na Java virtualnoj mašini (JVM) što omogućava veliku prenosivost (*engl. portability*) programa pisanih u Javi jer je samo potrebno prilagoditi JVM na novu platformu da bi na njoj radili postojeći programi.

Jednako kao što na ostalim (ne virtualnim) platformama programeri koriste mnogo različitih programskih jezika takva se potreba pojavila i na Java platformi. Programski jezici se razlikuju u pogodnosti za rješavanje različitih problema, neki imaju statički sustav tipova dok neki imaju dinamički, neki su lijeni (*engl. lazy* - podržavaju izvođenje koda samo kada se za tim javi potreba), itd. Jezici mogu biti imperativni, deklarativni, objektno orijentirani, funkcionalni ili mogu biti neka kombinacija ovih svojstava. Veliki razlog za postojanje mnogo programskih jezika je što je u različitim situacijama često u jednom programskom jeziku puno lakše riješiti određeni problem nego u nekom drugom.

Po mnogima dosta velik nedostatak Jave je to što je proces razvoja specifikacije i implementacije samog jezika jako spor. To znači da Java teško može pratiti moderne trendove u programskim jezicima te u tom pogledu zaostaje. Najveći razlog za spori razvoj je to što nove verzije Jave većinom zadržavaju kompatibilnost prema starom kodu i prema starom JVM-u što je bitno za jezik koji je tako puno korišten u produkcijskim sustavima kao Java. Veliki sustavi pisani u Javi ovise upravo o toj stabilnosti jezika i platforme.

Novi projekti ili manji projekti (npr. interni projekti za korištenje unutar tvrtke) često mogu eksperimentirati s alternativnim jezikom i postići brži i jednostavniji razvoj, smanjiti količinu koda koju je potrebno napisati za rješavanje određenih problema te olakšavati održavanje i promjene. Ovakav eksperiment može pokazati da li je prelazak na neki drugi jezik realna mogućnost i na većim projektima.

Trenutno na Java platformi postoji mnogo programskih jezika, od jezika-igračaka (*engl. toy language*) nastalih kao hobi projekt, do programskih jezika spremnih za korištenje u produkcijskim sustavima.

Neki jezici na java platformi:

- AspectJ - aspect-oriented ekstenzija Jave
- ColdFusion - skriptni jezik
- Groovy
- Scala
- Clojure - dijalekt Lisp-a
- JavaFX Script
- JRuby
- Jython
- Rhino
- itd.

2. SCALA

Scala je programski jezik koji podržava više paradigmi i integrira svojstva objektno orijentiranih i funkcionalnih programskih jezika. Kao što i samo ime jezika govori radi se o skalabilnom jeziku, što znači da se lagano prilagođava potrebama korisnika i veličini projekta. Scala daje naglasak povećavanju produktivnosti i iskustvo je pokazalo da se količina programskog koda prilikom prevođenja iz Jave u Scalu smanji u prosjeku za dva do tri puta. Već to pokazuje na jednu prednost u odnosu na Javu jer manja količina programskog koda znači i manje mogućnosti za pojavljivanje grešaka.

Programe pisane u Scali moguće je izvršavati na Java platformi i kompatibilni su s postojećim programima i bibliotekama pisanim u Javi. Postoji i implementacija Scala prevodilac koja proizvodi programe koje je moguće izvršiti na Microsoftvoj .NET platformi. Scala prevodilac na temelju Scala izvornog koda proizvodi *byte code* koji je gotovo identičan byte codu koji proizvodi Java prevodilac uz razliku da Scala programi dodatno koriste standardnu biblioteku Scala jezika: scala-library.jar.

Scalin prevodilac je napisao Martin Odersky koji je i autor Javinog referentnog prevodioca te koautor Javine implementacije generičkih tipova (*engl. generics*).

2.1. Objektno orijentirano programiranje

Scala je čisti objektno orijentirani programski jezik jer je sve u Scali objekt. Po tome se razlikuje od Jave u kojoj primitivni tipovi (boolean, int, long, float ...) nisu objekti. Tipovi podataka se definiraju korištenjem klasa i traitova (*engl. trait*) koji su moćnija varijanta sučelja (*engl. interface*) iz Jave jer se u njima može pojavljivati implementacija metoda te varijable vezane uz tu implementaciju.

2.2. Funkcionalno programiranje

Jedna od najvećih razlika između Scale i Jave je u tome što Scala podržava funkcionalno programiranje. Koristeći jednostavnu sintaksu omogućuje korištenje anonimnih funkcija, podržava funkcije višeg stupnja, ugnježđivanje funkcija, korištenje algebarskih tipove podataka i *pattern matching*. Funkcionalno programiranje je programska paradigma u kojoj se sve izvodi evaluiranjem matematičkih funkcija i izbjegava se korištenje stanja i promjenjivih (*engl. mutable*) podataka. Dok je u programima pisanim u imperativnom stilu naglasak na mijenjanju stanja objekata sa kojima se radi, programi pisani u funkcionalnom stilu daju naglasak na primjeni funkcija na objekte pri čemu one ne mijenjanju stanje originalnih objekata nego vraćaju novo stanje - nove objekte.

Jedno od najvažnijih svojstava čistih programskih jezika je to da rezultat izvođenja funkcije ovisi samo o parametrima funkcije te ako se funkcija pozove više puta uvijek daje jednake rezultate. Po ovome se funkcijski programski jezici razlikuju od imperativnih programskih jezika u kojima često rezultat funkcije ovisi o globalnom stanju, stanju objekta, ulazu od korisnika i slično. Pokazuje se da pristup koji koriste funkcionalni programski jezici omogućuje (čak i kada se koristi u imperativnim programskim jezicima) lakše testiranje (jer je moguće funkciju testirati kao crnu kutiju sa uvijek poznatim preslikavanjem ulaza na izlaze) ali i lakše razumijevanje i predviđanje ponašanja nekog programa.

2.3. Statički sustav tipova

Jezik Scala ima statički sustav tipova puno moćniji od Javinog. Ovdje su samo nabrojane neke od mogućnosti koje pruža Scalin sustava tipova:

- generičko programiranje, konstruktori tipova i *type* paremetri, definiranje varijance parametarskih tipova (*engl. nonvariant, covariant, contravariant*)
- klase i abstraktni tipovi mogu biti članovi objekata

Scala prevodilac u većini slučajeva može zaključiti (*engl. type inference*) kojeg je tipa neki izraz na temelju njegovog korištenja pa najčešće nije potrebno eksplicitno navoditi tipove. U Javi je kod deklaracija potrebno često tip navoditi i na lijevoj i na desnoj strani deklaracije.

Primjer Java:

```
List<String> lista1 = new ArrayList<String>();
String[] lista2 = new String[]{"A", "B", "C", "D"};
List<Map<String, Object>> listaMapa = pozivMetodeKojaVracaListuMapa();
```

Primjer Scala:

```
var lista1 = List()
var lista2 = Array("A", "B", "C", "D")
var listaMapa = pozivMetodeKojaVracaListuMapa
```

Primjer klase Osoba u Javi:

```
public class Osoba {

  private String ime;
  private String prezime;
  private Date datumRodjenja;

  public Osoba(String ime, String prezime, Date datumRodjenja) {
    this.ime = ime;
    this.prezime = prezime;
    this.datumRodjenja = datumRodjenja;
  }

  public String getIme() {
    return ime;
  }

  public void setIme(String ime) {
    this.ime = ime;
  }

  public String getPrezime() {
    return prezime;
  }

  public void setPrezime(String prezime) {
    this.prezime = prezime;
  }

  public Date getDatumRodjenja() {
    return datumRodjenja;
  }

  public void setDatumRodjenja(Date datumRodjenja) {
    this.datumRodjenja = datumRodjenja;
  }

  //nedostaje implemntacije metoda:
  //toString
  //equals
  //hashCode
}
```

Primjer klase Osoba u Scali:

```
case class Osoba(ime: String, prezime: String, datumRodjenja: String)
```

Gornji primjer dijela scala koda definira klasu sa javnim propertijima ime, prezime i datumRodjena koji su nepromjenjivi (*engl. immutable*). Moguća je i varijacije na temu u smislu da pojedini propertiji mogu biti i promjenjivi (*engl. mutable*) ako se pred njih doda ključna riječ `var`. Primjer nepromjenjive klase postoji u jeziku Java i to je String klasa, gdje objekti ove klase se nakon što su kreirani nikada ne

mijenjaju. Tako u slijedećem primjeru poziv metode na objektu tipa String nikad ne mijenja objekt nad kojim je metoda pozvana nego se kreira novi objekt na temelju starog.

```
String s = "ABC";
s.toLowerCase(); //s je i dalje "ABC"
```

U Scali se svaka varijabla može definirati kao muttable ili immutable.

```
val maxValue = 100 //nepromjenjiva vrijednost
var currentValue = 1 //vrijednost koju je moguće mijenjati
```

I u Javi je moguće varijable deklarirati nepromjenjivim koristeći ključnu riječ `final`, ali se ne koristi tako često kao u Scali. Prednost korištenja nepromjenjivih varijabli i objekata je u tome što se često mogu postići bolje performanse jer prevodilac može izvršiti neke optimizacije ako unaprijed zna da se objekt neće mijenjati (izvršavanje koda na više jezgri, lijeno izvršavanje, itd.).

Na slijedećih par primjera prikazane su samo neke stvari koje se mogu u Scali napraviti u jednoj kratkoj liniji koda dok bi u Javi odgovarajući kod bio nekoliko redaka i daleko teži za razumjeti na prvi pogled.

```
val o1 = Osoba("Ivan", "Senji", "1982-03-29")
val o2 = Osoba("Mario", "Popović", "1982-08-26")
val o3 = Osoba("Mario", "Marić", "1981-01-01")
val o4 = Osoba("Hrvoje", "Horvat", "1972-04-21")

val osobe = List(o1, o2, o3, o4)

// IMENA SVIH OSOBA
val imena = osobe.map(o => o.ime).toSet
//imena: Set(Ivan, Mario, Hrvoje)

// GODINE ROĐENJA SVIH OSOBA
val godine = osobe.map( o => o.datumRodjenja.substring(0,4) ).toSet
//godine: Set(1982, 1981, 1972)

// SVE GODINE OSIM 1982
godine - "1982"
//rezultat: Set[java.lang.String] = Set(1981, 1972)

// SVE OSOBE OSIM ONIH ROĐENIH 1982
osobe.remove( o => o.datumRodjenja.substring(0,4) == "1982" )
//rezultat: List(
//            Osoba(Mario,Marić,1981-01-01),
//            Osoba(Hrvoje,Horvat,1972-04-21)
//          )

//OSOBE PODIJELJENE U DVIJE GRUPE, ROĐENE PRIJE I POSLIJE 1982
osobe.partition( o => o.datumRodjenja.substring(0,4).toInt < 1982 )
//rezultat:   List(
//                 Osoba(Mario,Marić,1981-01-01),
//                 Osoba(Hrvoje,Horvat,1972-04-21)),
//            List(Osoba(Ivan,Senji,1982-03-29),
//                 Osoba(Mario,Popović,1982-08-26))

osobe.foreach( o => o match {
    case Osoba("Mario", _, datum)
      => println("Osoba se zove Mario i rođenja je datuma " + datum)
    case _ => println("Osoba se ne zove Mario")
})
```

```
//rezultat:
//Osoba se ne zove Mario
//Osoba se zove Mario i rođenja je datuma 1982-08-26
//Osoba se zove Mario i rođenja je datuma 1981-01-01
//Osoba se ne zove Mario
```

Samo neka svojstva jezika Scala su ilustrirana slijedećim primjerima:

Inferencija tipova	`var x = 8 //x je tipa Int`
	`x = "abc" //type mismatch: found String("abc")`
	`required: Int`

Hello World primjer	`object HelloWorld {`
	`  def main(args: Array[String]) {`
	`  println("Hello World!")`
	`  }`
	`}`

Funkcionalni koncepti - operacije se prevode u poziv metoda	`1 + 3 - 6` `1.+(3).-(6)`

Ugnježđivanje funkcija	`def function1(x : Int) = {`
	`  def function2() = {`
	`  println(x)`
	`  }`
	`  function2()`
	`}`

First-class funkcije	Funkcije mogu biti parametri drugim funkcijama:
	
	`def foo(funkcija: () => Unit) : Unit = {`
	`  while(true) {`
	`  funkcija()`
	`  Thread.sleep(1000)`
	`  }`
	`}`

Funkcije kao parametri funkcijama iz standardne biblioteke	`val brojevi = List(3,6,2,4,1,8,7,9,10)`
	`brojevi.foreach( x => println(x) )`
	`brojevi.sort( (x,y) => x < y )`
	`//rezultat: List(1, 2, 3, 4, 6, 7, 8, 9, 10)`
	`(brojevi.min, brojevi.max)`
	`//rezultat: (1,10)`
	`brojevi.map( x => x*x*x )`
	`//rezultat: List(27, 216, 8, 64, 1, 512, 343, 729, 1000)`
	`brojevi.filter( x => x < 5 )`
	`//rezultat: List(3, 2, 4, 1)`
	`brojevi.sum //suma na 1 način`
	`//rezultat: 50`
	`brojevi.foldLeft(0)( (x,y) => x + y ) // suma na 2 način`
	`//rezultat: 50`
	`brojevi.foldLeft(1)( (x,y) => x * y ) //umnožak`
	`//rezultat: 725760`
	`brojevi.partition( broj => broj < 3 )`
	`//rezultat: (List(2, 1),List(3, 6, 4, 8, 7, 9, 10))`
	`brojevi.zipWithIndex`
	`//rezultat: List((3,0), (6,1), (2,2), (4,3), (1,4), (8,5), (7,6), (9,7), (10,8))`

Djelomično primijenjene funkcije	`def zbroji(x : Int, y : Int, z : Int) = x + y + z`
	`val zbrojis10i100 = zbroji(10, 100, _ : Int)`
	`zbrojis10i100(2) // vraća 112`

Rad s regularnim izrazima	`val mob = "(\\d{3}) (\\d{4}) (\\d{3})".r`
	
	`def zamjeni(broj: String) = broj match {`
	`  case mob(a,b,c) => c + b + a`
	`  case _ => "Krivi format broja"`
	`}`
	
	`zamjeni("123 4567 890")`
	`//rezultat: java.lang.String = 8904567123`
	
	`zamjeni("123 456 789")`
	`//rezultat: Krivi format broja`

Rad s xml-om	`val xml = <osobe>{`
	`  osobe.map( o =>`
	`  <osoba>`
	`  <ime>{o.ime}</ime>`
	`  <prezime>{o.prezime}</prezime>`
	`  </osoba>)`
	`  }</osobe>`
	`//rezultat:`
	`//<osobe>`
	`//  <osoba><ime>Ivan</ime><prezime>Senji</prezime></osoba>`
	`//  <osoba><ime>Mario</ime><prezime>Popović</prezime></osoba>`
	`//  <osoba><ime>Mario</ime><prezime>Marić</prezime></osoba>`
	`//  <osoba><ime>Hrvoje</ime><prezime>Horvat</prezime></osoba>`
	`//</osobe>`
	
	`xml \\ "ime"`
	`//rezultat: NodeSeq(<ime>Ivan</ime>, <ime>Mario</ime>,`
	`<ime>Mario</ime>, <ime>Hrvoje</ime>)`
	
	`(xml \\ "ime").text`
	`//rezultat: IvanMarioMarioHrvoje`

2.4. Scala biblioteke

Scala se u sintaksi razlikuje dosta od Jave i ima ugrađene mogućnosti za kreiranje DSL-ova (*engl. Domain Specific Language*), pod-jezika unutar Scale koji rješavaju probleme u nekoj užoj domeni. Te mogućnosti koriste mnoge biblioteke pisane u Scali. Među češće korištenima su biblioteke za testiranje pisane u Scali kao što su Scalatest, scalacheck i specs.

2.4.1. Scalatest

Korištenje ove biblioteke za testiranje prikazano je najbolje na slijedećem primjeru na kojem se vidi da je postignuta velika čitljivost koda. Ovaj komad koda može čitati skoro kao specifikacija pisana na engleskom jeziku.

```
class StackSpec extends FlatSpec with ShouldMatchers {
  "A Stack" should "pop values in last-in-first-out order" in {
    val stack = new Stack[Int]
    stack.push(1)
    stack.push(2)
```

```
    stack.pop() should equal (2)
    stack.pop() should equal (1)
  }

  it should "throw NoSuchElementException if an empty stack is popped" in {
    val emptyStack = new Stack[String]
    evaluating { emptyStack.pop() } should produce [NoSuchElementException]
  }
}
```

2.4.2. Lift

Lift je framework za izradu web aplikacija pisan u Scali koji se hvali jednostavnošću izrade aplikacija kojom se također hvali i popularni Ruby on Rails. Prednost Lift-a je što se u njemu mogu koristiti sve postojeće Java biblioteke te aplikacije se mogu pokrenuti na bilo kojem aplikacijskom serveru (Tomcat, Jetty, Resin, Glassfish i drugima). Prezentacijski sloj koristi predloške bazirane na standardnom XHTML formatu i za razvoj je moguće koristiti postojeće Java razvojne alate (Eclipse, Netbeans, IDEA). Lift aplikacije podržavaju napredne web tehnologije kao što su Ajax i Comet koje omogućuju jednostavnu izradu interaktivnih aplikacija. Comet je model web aplikacije u kojem se pomoću dugotrajnih HTTP requesta prema serveru omogućuje da server šalje podatke push pristupom prema web pregledniku.

Ekspresivnost i jednostavnost kojom je moguće postići neke složene interakcije s korisnikom najbolje će prikazati slijedeći primjer potpuno funkcionalne aplikacije za chat u samo tridesetak linija koda.

```
object ChatServer extends LiftActor with ListenerManager {
  private var messages = List("Welcome")
  def createUpdate = messages
  override def lowPriority = {
    case s: String =>
      messages ::= s
      updateListeners
  }
}
class Chat extends CometActor with CometListener {
  private var msgs: List[String] = Nil
  def registerWith = ChatServer
  override def lowPriority = {
    case m: List[String] =>
      msgs = m
      reRender(false)
  }
  def render =
    <div>
    <ul> { msgs.reverse.map(m => <li>{m}</li>) } </ul>
    <lift:form> { SHtml.text("", s => ChatServer ! s) }
      <input type="submit" value="Chat"/>
      </lift:form>
    </div>
}
```

Jedna od stvari kojima se Lift hvali (osim elegancije koda) je da je po većini istraživanja pronađeno da je Lift među najsigurnijim web frameworcima. Lift postiže sigurnost dobrim defaultnim opcijama tako da su aplikacije i bez posebne intervencije i kodiranja sigurne.

3. GROOVY

Groovy je dinamički programski jezik čija je ciljana platforma JVM. On se dinamički prevodi u JVM bytecode, te se može koristiti sa ostalim Java kodom i bibliotekama.

Sintaksa Groovy-ja i Jave je vrlo slična tako da Java programeri mogu vrlo brzo svladati korištenje Groovy sintakse.

Groovy za razliku od Jave omogućuje dinamičke tipove (tj. nije strogo tipiziran), closure, nadjačavanje operatora, DSL (*engl. Domain Specific Language)*, ima ugrađenu podršku za regularne izraze, omogućuje jednostavniji rad sa listama i mapama, itd.

Samo neka svojstva jezika Groovy su ilustrirana slijedećim primjerima:

Hello World primjer	`println "Hello World!"`

Closure
- slično Java inner klasi, samo što je to jedna metoda koja se može pozvati sa proizvoljnim argumentima (ako nema navedenog parametra defaultno ime za parametar je "it")
	`def closure = { param -> println("hello ${param}") }`
	`closure.call("world!")`
	
	`def closure = { println "hello " + it }`
	`closure.call("world!")`
	
	`def words = ['HROUG', '15', 'Rovinj', '2010']`
	`words = words.findAll{ w -> w.size() >= 4}`
	`words.each{it -> println '$it'}`

Primjer za regularne izraze	`def mobPattern = /(\d{3}) (\d{4}) (\d{3})/`
	
	`def drugiOperater = "097 1234 093".replaceAll(mobPattern) {mob, mobStart, mobMid, mobEnd ->`
	`  return "$mobEnd $mobMid $mobStart"`
	`}`

Kao i ranije navesti ćemo par primjera koji prikazuju mogućnosti Groovy-a, te njegove prednosti nad Javom.

```
class Osoba {
    String ime
    String prezime
    String datumRodenja
}

def osoba1 = new Osoba(ime:"Ivan",   prezime:"Senji",
datumRodenja:"1982-03-29")
def osoba2 = new Osoba(ime:"Mario",  prezime:"Popović", datumRodenja:"1982-
08-26")
def osoba3 = new Osoba(ime:"Mario",  prezime:"Marić",   datumRodenja:"1981-
01-01")
def osoba4 = new Osoba(ime:"Hrvoje", prezime:"Horvat", datumRodenja:"1972-
04-21")

def osobe = [osoba1,osoba2,osoba3,osoba4]

// IMENA SVIH OSOBA
def imena = osobe.collect({it -> it.ime}).toSet()
//imena: Set(Ivan, Mario, Hrvoje)
```

```
// OSOBE ROĐENE GODINE RAZLIČITE OD 1982
def listGodinaRodenja = osobe.findAll{
                        it -> it.datumRodenja.substring(0,4) != '1982'}

//OSOBE PODIJELJENE U DVIJE GRUPE, ROĐENE PRIJE I POSLIJE 1982
def listSplit = osobe.split{
                it -> it.datumRodenja.substring(0,4).toInteger() < 1982}
```

3.1. Grails

Grails je web framework koji se temelji na Groovy-u i njegovim dinamičkim svojstvima. Grails su izgrađeni na principu "Nemoj se ponavljati" (eng. Don't Repeat Yourself) te time dosta smanjuju kompleksnost izgradnje web aplikacije na JVM. Grails je modularni framework, te pokušava kroz core tehnologije i pripadne module riješiti i ubrzati što više dijelova web razvoja .

Sljedeći primjer prikazuje kako je uz pomoć grails moguće sa samo nekoliko naredbi u komandnoj liniji i promjena u kodu dobiti jednostavu HelloWorld web aplikaciju.

```
// ove naredbe izvršavaju se u komandnoj liniji
grails create-app helloworld
grails create-controller hello

class HelloController {
  def world = {
// definiranje teksta za ispis
    render "Hello World!"
  }
}

// ove naredba izvršava se u komandnoj liniji
grails run-app
```

4. CLOJURE

Clojure je dinamički programski jezik čija je ciljana platforma JVM te se izvorni kod prevodi u JVM bytecode i ima jako dobru podršku za pozivanje postojećeg Java koda i za višedretveno programiranje. Clojure je dijalekt programskog jezika Lisp što znači da je prvenstveno funkcionalan programski jezik u kojem se programski kod može tretirati kao podaci te time podržava napredan sustav makroa.

Clojure pruža implementaciju STM (*engl. Software Transactional Memory*) za postizanje konkurentnosti. STM je model za kontrolu konkurentnosti sličan po ideji transakcijama na bazi i alternativa je sinkronizaciji baziranoj na zaključavanju. Transakcija je u kontekstu STM-a komad programskog koda koji piše i čita po dijeljenoj memoriji. Prednost STM u odnosu na druge metode sinkronizacije je što je to optimistična metoda i omogućuje veću razinu konkurentnosti jer nema čekanja na pristup resursima. Danas je STM aktivna tema istraživanja kao alternativna tehnika za postizanje sinkronizacije.

U slijedećoj tablici je samo nekoliko primjera jednostavnih izraza pisanih u jeziku Clojure.

Lista	`(a b c)`
Vektor	`[1 2 3]`
Mapa	`{:a 1 :b 2}`

Zbrajanje	`(+ 1 2 3)`
Ispisivanje	`(println "Hello")`
Pozivanje funkcije	`(ime-funkcije arg1 arg2 arg3)`
Definicija funkcije	`(defn hello [name]` `  (println "Hello," name))`

5. OSTALI JEZICI

5.1. JRuby

JRuby je implementacija programskog jezika Ruby u Javi. Glavna karakteristika JRubyja je dobra integracija s Javom koja omogućuje integraciju JRuby interpretera u bilo koju Java aplikaciju s komunikacijom koda u oba smjera.

JRuby implementacija Ruby jezika je u početku bila sporija od referentne implementacije jezika dok je u današnje vrijeme i nekoliko puta brža (u prosjeku oko 2.5 puta). JRuby programi se mogu izvršavati u interpretiranom načinu, AOT (*engl. ahead-of-time*) načinu i JIT (*engl. just-in time*) načinu. Razlika između AOT i JIT načina je da se u AOT načinu program prevodi u sistemski binarni kod prije izvođenja programa, dok se prevođenje kod JIT načina obavlja za vrijeme izvođenja programa.

Ruby na kojem se JRuby bazira je dinamički i reflektivan programski jezik opće namjene (kreiran u Japanu) koji je u sintaksi inspiriran mješavinom Perl-a i Smaltalka. Ruby podržava više programskih paradigmi: funkcionalnu, objektno-orijentiranu, imperativnu i reflektivnu.

Hello World	`puts "Hello World!"`

Liste	`a = [1, 2, [4, 5], [6, "sedam"]]`
	`a.flatten # => [1, 2, 4, 5, 6, "sedam"]`
	`a.reverse # => [[6, "sedam"], [4, 5], 2, 1]`

Iteriranje	`[1, 'xyz', 3.141].each { |elem| puts elem }`
	`# 1`
	`# xyz`
	`# 3.141`
	
	`a.each_index { |i| puts "#{i} : #{a[i]}" }`
	`# 0 : 1`
	`# 1 : 2`
	`# 2 : 45`
	`# 3 : 6sedam`

5.1.1. Ruby on Rails

Jedan od najvećih razloga za korištenje JRubyja je *Ruby on Rails*. Ruby on Rails je web framework pisan u Rubyju (koji je po mnogima proslavio Ruby) kojeg najbolje opisuje slogan s web stranice projekta: Ruby on Rails je opensource web framework optimiziran za sreću programera i održivu produktivnost koji omogućuje pisanje lijepog koda. Postojeći korisnici Ruby on Rails frameworka (ali i oni koji si tek zainteresirani za njegove korištenje) prelaskom na JRuby dobivaju bolje performasne na Java platformi uz zadržavanje elegantnosti programiranja u Rubyju i dodatno se otvara mogućnost korištenja postojećih Java librarija i postojećeg koda pisanog u Javi ili nekom drugom JVM jeziku.

5.2. Jython

Jython je implementacija programskog jezika Python pisana u Javi i koja se izvodi na JVM. Jython programi mogu bez problema koristiti sve postojeće Java klase.

Python je programski jezik koji daje naglasak čitljivosti koda i pruža velike mogućnosti uz jednostavnu sintaksu i opsežnu standardnu biblioteku. Na prvi pogled ga od C porodice jezika najviše razlikuje korištenje indentacije (uvlačenja) za delimitiranje blokova koda i programskih struktura. Python podržava objektno-orijentirano, imperativno i jednim dijelom i funkcionalno programiranje, ima dinamički sustav tipova i automatski rad s memorijom. Zbog tih njegovih svojstava izuzetno je pogodan kao skriptni jezik.

Jython uz sve prednosti koje Python nudi kao programski jezik, nudi i interoperabilnost s Javom. Moguće je koristiti sve postojeće Java biblioteke koji pokrivaju veliko područje riješenih problema.

Neke mogućnosti Jythona prikazane su na primjerima u slijedećoj tablici.

Hello World	`print "Hello, world!"`

Čitanje iz datoteke	`file = open("Test.txt", "r")`
	
	`for line in file.readlines():`
	`  print line`
	
	`file.close()`

Korištenje Javinog Swing GUI librarija	`from javax.swing import *`
	`frame = JFrame("Hello Jython")`
	`label = JLabel("Hello Jython!", JLabel.CENTER)`
	`frame.add(label)`
	`frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)`
	`frame.setSize(300, 300)`
	`frame.show()`

Objektno orijentirano programiranje	`class Hello:`
	`  def __init__(self, name="John Doe"):`
	`  self.name = name`
	
	`  def greeting(self):`
	`  print "Hello, %s" % self.name`
	
	`pero = Hello("Pero Perić")`
	`pero.greeting()`

6. ZAKLJUČAK

Koristeći alternativni jezik na Java platformi moguće je postići znatno smanjivanje programskog koda kojeg je potrebno napisati te time skratiti vrijeme razvoja aplikacije te više vremena posvetiti implementaciji poslovne logike i poslovnih pravila. Gotovo jedini nedostatak s kojim smo se susreli je to da je podrška za alternativne jezike relativno slabija u razvojnim alatima (Eclipse, Netbeans i IntelliJ IDEA), ali postoje naznake da će se razvojni alati poboljšati.