Scala

la posteulo por la Java-trono.

Moderna programlingvo sur amaso-platformo

Christoph Knabe

Beuth-Hochschule Berlin

http://public.beuth-hochschule.de/~knabe

Enhavo

- Enkonduko
- Kiel Java, sed pli simpla
- Stirstrukturoj, Trajtoj
- Java-kongrueco + Demo
- Struktura komparo kaj elprenado
- Funkciaj kapabloj + Demo, Anaso-tipsistemo
- Gastigo de Uzkamp-Specifaj-Lingvoj (USL)
- Paraleco + Demo, Skalebleco
- Komparo kun Aspektema Programado (AOP)
- La TTT-kadrosistemo Lift: Demo
- Konkludo

Enkonduko: Tempo por novo

- Java ĉe ni uzata de 1998
- Aplikaĵoprogramado ankoraŭ pena
  - Sensangaj aplikokampo-modelklasoj (“entities”)
  - Logiko-servoj apartigitaj de la datumoj
  - Kompleksaj uzo-kadrosistemoj (“UI frameworks”)
- Leviĝanta konkurenco
  - Ruby on Rails (TTT-kadrosistemo)
  - C# 3.0 kun funkciaj aldonoj (i.a. LINQ), F#

Enkonduko: Kiu estas malantaŭ Scala?

- Martin Odersky
  - Doktoriĝis ĉe N. Wirth
  - Aŭtoro de pizza, GJ
  - Aŭtoro de la Java5-tradukilo
  - Prof. @
    Programming Methods Group (LAMP)
    IC, EPFL Lausanne, Schweiz
- LangPop Rang 28
  - Discussion Rang 16
  - Timeline: Alta de 2009

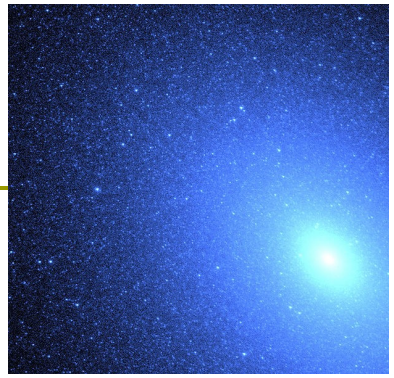
Enkonduko: Scala-superrigardo

- Scala: ”SCAlable LAnguage“ (SKAlebla Lingvo)
- Plene objektema (“object oriented”)
- Plene Java-kongrua
- Plene funkcia
- Statika tipsistemo
- Senredunda, orteca sintakso
- Erlang-agantoj (“actors”) por alta paraleco
- Enkonstruita XML-kapablo
- USL-taŭga (USL=uzkamp-specifa-lingvo, “DSL”)

Enkonduko: Scala kiel minilingvo (“scripting language”)

- Kiel komandinvito (“command prompt”):
  - *scala*↵
    scala> ***val*** *r = 1*↵
    scala> *println(2*Math.Pi*r*r)*↵
    6.283185307179586
    scala>
- Kiel komando-datumaro sen ĉirkaŭa klaso:
  - dir.scala:
    ```
    val dir = new java.io.File(".")
    val files = dir.list
    files foreach println
    ```
  - *scala dir.scala*↵
    hello.scala
    printargs.scala
    pa.scala
    paramwithtypes.scala
    ...

Java, sed pli simpla 1



- Ĉiu valoro estas objekto:
  - scala> *1 .+(2)*↵
    res0: Int = 3
    scala> *1+2*↵
    res1: Int = 3
  - Scala-valortipoj ŝpareme realiĝas per primitivaj Javo-tipoj
- Ĉiu operacisimbolo estas metodo:
  - En klaso BigInt:
    def + (that: BigInt): BigInt =
    new BigInt(**this**.bigInteger.add(that.bigInteger))
- Senrompa skribformo por legado (“uniform access principle”)
  (ĉu konstanto, variablo, senparametra funkcio):
  - scala> ***val*** *n = list.size*
    n: Int = 1

Java, sed pli simpla 2

- Tipkonkludo, generaj parametroj per rektaj krampoj **[…]** :
  - scala> ***val*** *map1 =* ***new*** *java.util.HashMap[String,Int]*↵
    map1: java.util.HashMap[String,Int] = {}
  - scala> ***val*** *map2: java.util.Map[String,Int] =* ↵
    | ***new*** *java.util.HashMap*↵
    map2: java.util.Map[String,Int] = {}

- Punktokomo-konkludado:
  - scala> ***val*** *a = 1*↵
    a: Int = 1
    scala> ***val*** *b = (a*↵
    | *+ 3)*↵
    b: Int = 3

Java, sed pli simpla 3

- Klasoparametroj difinas konstruilon + uzorajtojn:
  - **class** MyClass(x: Int, **val** y: Int, **var** z: Int){ … }
    val a = **new** MyClass(1,2,3)
    a.x //Ĉiu ekstera uzo malpermesita
    val b = a.y
    a.y = 5 //Skribado malpermesita
    val c = a.z
    a.z = 9

Java, sed pli simpla 4

- Atribut-uzometodoj estas redifineblaj:
  - ```
    class Halfer {
      private var _n = 0;
      def n_= (n: Int){    // storilo ("setter")
        require(n%2 == 0, "Odd " + n)
        _n = n
      }
      def n = _n / 2   // legilo ("getter")
    }
    ```
  - Uzado:

    ```
    val h = new Halfer
    h.n = 7   //throws IllegalArgumentException
    h.n = 100
    println(h.n) // 50
    ```
    ```

Java, sed pli simpla 5

- Java: La klaso-objekto-rompo:
  - ```
    class Counter {
      private static int count = 0;
      static void add(final int n){ count += n; }
      static void sub(final int n){ count -= n; }
    }
    Counter.add(5);
    Counter.sub(2);
    ```
  - Ĉe kiom da lokoj en `Counter` modifenda por pluraj nombriloj?
- Scala: **object** difinas unuopul-klason ("Singleton"):
  - ```
    object Counter {
      private var count = 0
      def add(n: Int){ count += n }
      def sub(n: Int){ count -= n }
    }
    Counter add 5
    Counter sub 2
    ```

Java, sed pli simpla 6

- Java: Malsama sintakso por kondiĉaj partoj:
  - kondiĉa ordono:
    if(kondiĉo)
      jesOrdono
    else
      neOrdono
  - kondiĉa esprimo:
    kondiĉo **?** jesEsprimo **:** neEsprimo
- Scala: Unueca sintakso por kondiĉaj partoj:
  - **if** (kondiĉo) jesEsprimo **else** neEsprimo
  - Ekz-o: **if**(a<b){
      println(a)
    }**else**{
      println(b)
    }
  - Ekz-o: **val** min = **if**(a<b) a **else** b

Stirado: Ripetoj

- **while**-ripetoj kiel en Java
  - **while**(kondiĉo)
    esprimo
  - **do** esprimo **while**(kondiĉo)
- Neniu meztestanta ripeto
  - Sen **break**;
- **for**-esprimo: Simplaj ekzemploj:
  - **for**(i **<-** 1 to 10) println(i)
  - **for**(i **<-** 1 to 10 by 2) println(i)
  - Lasta ripeto ekzakte kongruas kun:

```
                                        //implica transformo:
val one: scala.runtime.RichInt = Predef.intWrapper(1)
val oneToTen: scala.Range.Inclusive = one.to(10)
val oneToTenByTwo: scala.Range = oneToTen.by(2)
for(i <- oneToTenByTwo) println(i)
```

Stirado: branĉigoj

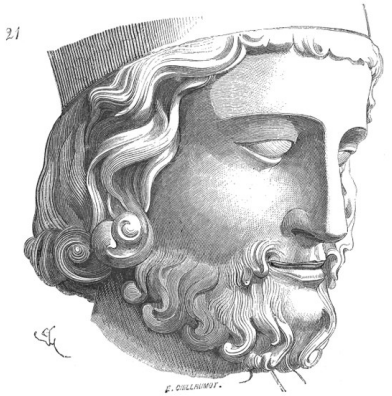
- **match**-esprimo:
  - Anstataŭigas **switch**, multe pli potenca, ekz-o:

```
val manĝo = if (args.isEmpty) "" else args(0)
val aldono = manĝo match {
  case "tomato" => "salo"
  case "fritoj" => "majonezo"
  case "ovoj" => "ŝinko"
  case _ => throw new IllegalArgumentException(manĝo)
}
println(aldono)
```

- Esceptotraktado:

```
try {
  doSomething()
} catch {
  case ex: java.io.FileNotFoundException =>
    println("Not found")
  case ex: Exception =>
    throw new RuntimeException("Cannot doSomething", ex)
}
```

Trajtoj



- La interfaco-dilemo en Java:
  - Malvasta interfaco: Malkomforta uzado
  - Vasta interfaco: Malkomfortaj realigoj
  - Ekz-o: Interfaco `java.lang.CharSequence` difinas nur 4 metodojn: `charAt`, `length`, `subSequence`, `toString`. Aliajn, ekz-e `trim` oni devus difini en ĉiuj realigantoj.
- *Trajto* estas parte realigita interfaco:
  - **trait** CharSequence { ...
    def subSequence(start: Int, end: Int): CharSequence
    def trim = {
    val start = indexOfFirstNonwhiteChar(**this**)
    val end = indexOfFirstTrailingWhiteChar(**this**)
    subSequence(start, end)
    }
    }

Java-kongrueco

- Scala tradukiĝas al Java-bajtokodo
  - Klasoj, metodoj, valortipoj, signoĉenoj, esceptoj:
    Rekta bildigo al la respondaj Java-konstruaĵoj
  - **object** MyObj {}
    estas bildigita jene, garantiante unu ekzempleron:
    final class MyObj$ **implements** scala.ScalaObject {
    static final MyObj$ MODULE$;
    MyObj$();
    }
  - Scala-trajto kreas Java-interfacon kun aldonoj.
    Abstrakta Scala-trajto iĝas ekzakte interfaco.
  - Java-interfaco povas realigxi per Scala-trajto/klaso.
  - Genereco: Preskaŭ 1:1-respondo (~ wildcards)
  - Prinotoj: Java-prinotoj uzeblaj en Scala , ekze-e `@Test`
  - Vidu demonstron `StoreScribble.scala`
    IDEA > File > Settings > Editor > Fonts > Scheme > BigPresentation

Struktura Komparo kaj Elprenado („Pattern Matching“) 1

- Struktura komparo ebligas tipsekuran elprenadon:
  - ```
    val numeroj = List[Float](1,2,3,4,5)
    val (sumo, averaĝo) = sumoKajAveraĝo(numeroj)
    println("Sumo: " + sumo + ", Averaĝo: " + averaĝo)
    ```
- Artikolo “Scala for Java Refugees”: Pattern Matching
  - Color-hierarkio: http://www.codecommit.com/blog/scala/scala-for-java-refugees-part-4
  - Ekzemplo el tio: `PatternMatching.scala`
  - Ebligas tipsekuran okazodistingon
  - Anstataŭigas **`instanceof`** kaj tipaserton („type cast“).
  - Taŭgas por inspektado de sintaksarbo laŭ la Interpretilo-skemo („`Interpreter` design pattern“)
    ```

Struktura Elprenado 2: Option

- Antaŭdifinita klaso `Option[T]` deklaras laŭvolan datumon de la tipo `T`:
  - ```
    class Person(
      val name: String, val eMail: Option[String]
    )
    ```
- **case**-klaso `Some[T]` **extends** `Option[T]` enhavas la datumon:
  - `new Person("Horst", Some("horst@berlin.de"))`
- **case**-klaso `None` **extends** `Option[T]` ne enhavas datumon:
  - `new Person("Horst", None)`
- Statike tipsekura elprenado de la laŭvola datumo per **match**-esprimo:
  - ```
    p.name + p.eMail match { case Some(x) => "," + x
                             case None => ""          }
    ```

Funkcia: Esprimo-parametroj

- Difino
  - **Esprimo-parametro** („call-by-name parameter“):
    La transdonita esprimo dum ĉiu uzo denove elkalkuliĝas
    (Kontraŭe Java: Nur valor- aŭ referenc-parametroj).
- Demo:
  - **var** v = 5
    def callByName(e: => Int) = **while**(v>0){ v = v-1
      println(e)
    }
    callByName(v+2) // 6 5 4 3 2
- Utila ekz-o protokolado:
  - **def** debug(msg: => String) =
      if(isDebugEnabled)println(msg) ...
    debug("Person " + p.name + " with id " + p.id)
  - Avantaĝo: La aktuala parametro elkalkuliĝas nur, se vere necesas.

Funkcioj unuarangaj valoroj

- Java: Metodoj duarangaj (kompare al objektoj)
  - difini √ voki √ literaĵo × stori × transdoni × liveri ×
- Funkcia literaĵo: $(x_1: T_1, x_2: T_2, \ldots) \Rightarrow$ *esprimo*$(x_1, x_2, \ldots)$
  - (n: Int) => n%2 == 0 //para numero
- Stori funkcion:
  - **val** even = (n: Int) => n%2 == 0
- Transdoni funkcion, ĉi tie al `filter` :
  - **val** numbers = 3 to 12
    val evenNumbers = numbers.`filter`(even)
    assert(evenNumbers === List(4, 6, 8, 10, 12))
- ("higher order")(Meta-)funkcio liveranta funkcion:
  - **val** dividableBy3 = (n: Int) => n%3 == 0
    def bothHold(f: Int => Boolean, g: Int => Boolean) =
    (x: Int) => f(x) && g(x)
    val numbers6 = numbers.filter(bothHold(even, dividableBy3))
    assert(numbers6 === List(6, 12))

Funkcia: Konciza stilo

- Ekz-o: Testi majusklan literon en Java:
  - ```
    public boolean hasUpperCase (final String s) {
        final int len = s.length();
        for (int i=0; i<len; i++){
            if (Character.isUpperCase(s.charAt(i))){
                return true;
            }
        }
        return false;
    }
    ```
  - Imperativa stilo: `i` estas variablo; ekzakta serĉmanier-priskribo
- Ekz-o: Testi majusklan literon en Scala:
  - **def** hasUpperCase(s: String) = s.exists(c => c.isUpper)
  - Funkcia stilo: Nur konstantaj valoroj
  - Serĉripeto taskigita al metodo `exists` el `Iterable[A]`:
    ```
    def exists(predicate: A => Boolean): Boolean
    ```
    ```

Funkcia: Propraj Stirkonstruaĵoj („control constructs“) difineblaj

- Problemo: Neniu **`break`** in Scala:
  - ⇒ **Neniu meztestanta ripeto.**
  - Tamen utila por ripetotipo:
    havigo, sukcesotesto, prilaborado
- Ekz-o Meztestanta ripeto mem difinita:

```
def loopGetTestProcess[Item]
   (getItem: => Item)                    // : => Prokrastita elkalkulado
   (isContinue: Item=>Boolean)
   (processItem: Item=>Unit)
{ ... }
```

- Ekz-o Meztestanta ripeto uzata:

```
def printLines(in: java.io.BufferedReader){
  loopGetTestProcess(in.readLine())(e=>e!=null){
    println
  }
}
```

Anaso-tipsistemo („duck typing“) statika: Strukturaj Tipoj

- Dinamike tipigitaj lingvoj (Ruby, Python):
  - Variablo ĉiam povas ŝanĝi sian objekttipon.
  - “Se kvakas kiel anaso kaj paŝas kiel anaso, ĝi estas anaso.”
  - Tasko: Se `close`-metodo ekzistas, ĝin aŭtomate voki.
- Java 6: close() en multaj klasoj
  - java.io.Reader, java.sql.Connection
  - Auto-Close ne ĝenerale formulebla.
- Scala: Strukturaj tipoj
  -
```
def use(resource: { def close() }, action: => Unit){
  try{
    action;
  }finally{
    resource.close()
  }
}
```

Gastiga lingvo por USLoj -

USL = uzkamp-specifa-lingvo (“DSL = Domain Specific Language”)

- Ekz-e Gramatiko de aritmetika esprimo:
  - expr ::= term {'+' term | '-' term}.
    term ::= factor {'*' factor | '/' factor}.
    factor ::= floatingPointNumber | '(' expr ')'.
- En Scala kiel USL:
  - ```
    import scala.util.parsing.combinator._
    class Arith extends JavaTokenParsers {
      def expr: Parser[Any] = term ~ rep("+"~term | "-"~term)
      def term: Parser[Any] = factor ~ rep("*"~factor | "/"~factor)
      def factor: Parser[Any] = floatingPointNumber | "("~expr~")"
    }
    ```
  - Klarigo:

`~`	(tildo)	Sinsekvo
`|`	(vertikala streko)	Alternativo
`rep`	(“repeated”)	Ripeto
`opt`	(“optional”)	Laŭvoleco

  - Vidu objekton `ArithParse` en programredaktilo.
    ```

Menu-difino en Lift

- Permana konstruado en Lift 1:
  - **def** mainMenu: List[Menu] = List(...
    Menu(Loc("newsMenuLoc", List("news", "index"),
    "Novaĵoj", mainGroup)),
    Menu(Loc("subscriptionManageLoc",
    List("subscriptions", "manage"),
    "Abonoj" , IfLoggedIn, mainGroup))
)
- En Lift 2 kiel USL:
  - **def** mainMenu: List[Menu] = List(...
    Menu("newsMenuLoc", "Novaĵoj") / "news" / "index"
    >> mainGroup,
    Menu("subscriptionManageLoc", "Abonoj")
    / "subscriptions" / "manage"
    >> IfLoggedIn >> mainGroup
)
    Vidu `mainMenu` en klaso `bootstrap.liftweb.Boot`

Agiloj por grandskala paraleleco („actor model“)

- Samtempaj Java-stirfadenoj (“threads”) kun riskoj:
  - Konkura hazardo (hazarda rezulto depende de tempobezono)
  - Morta blokiĝo (eterna reciproka atendado de paralelaj procezoj)
  - Tro storkonsuma por grandskala paraleleco (>5000 fadenoj) Java taŭgas por milionoj da objektoj, sed nur miloj da fadenoj.
  - Tro tempokonsuma por oftaj fadeno-ŝanĝoj
- Scala: Samtempaj agiloj en Erlang-stilo:
  - Neniu komuna storo ⇒ Neniu tempa kunordigo de aliro
  - Peto/Respondo tempe sendependa per mesaĝo-sendado
  - Nur konstantajn valorojn aŭ agilo-adresojn sendi
  - Ankaŭ kun miloj da agiloj bezonas nur 1 fadenon po CPU
- Sintakso:
  - Sendi: agilo ! mesaĝo
  - Ricevi: `react` { **`case`** *pattern* => *action*; **...** }
  - Demo: liftbuchcode → mvn jetty:run → knabe@beuth sendi

Agiloj: Ekzemplo

- Taksotesto (“benchmark”):
  - Produktanto-konsumanto-ĉeno el *n* fadenoj | *n* agiloj
  - Mesaĝon *n*-foje sendi $\Rightarrow$ O(n^2)
  - En Java kiel ĉeno fadeno-bufro-fadeno-bufro ...
  - En Scala kiel ĉeno agilo-mesaĝo-agilo-mesaĝo ...
- Testo-rezultatoj:
  -

N	Java [ms]	Scala [ms]
10	0	16
100	47	109
1000	4407	3562
2000	19250	17250
3000	44953	36046
4000	80360	72063
5000	125485	109609
6000	OutOfMemoryError: Unable to create native thread	162391

Skalebleco: Hierarkiaj pakaĵoj

- Hierarkiaj pakaĵoj:
  - Genera videbleco-modifanto **`private[`***Component***`]`**
  - *Component* povas esti pakaĵo aŭ klaso.
  - **`private[`**`a.b`**`]`** permesas aliron el `a.b`, `a.b.c` ktp., sed ne el `a.`*x*.
  - Ekz-o: Pakaĵoj `sample.math`, `sample.math.impl`,
    Programo `sample.MathSample`
- Skalebleco en Scala:
  - De komando-sekvo ĝis vere hierarkiaj pakaĵoj
  - De sinsekva ĝis multnombre paralela
  - Pligrandigebla per propraj stirstrukturoj
  - Pligrandigebla al USL-oj (uzkamp-specifaj-lingvoj)

Komparo al AOP

- Aspekt-Orientita Programado (AspectJ)
  - Potenca tekniko
  - Tuj en Java-projektoj uzebla
  - Klara labordivido:
    Recepto-farantoj skribas aspektojn.
    Rezepte-uzantoj skribas Javo-kodon.
- Scala
  - Lernbezono pli alta
  - Produktiveco-potencialo pli alta
  - Ebligas novspecajn kadrosistemojn.

La TTT-kadrosistemo **Lift**, baziĝanta sur Scala

- Sintezo de la plej bonaj konceptoj
  - Dialogo-ŝablonado kiel en Wicket
  - Facilega enmetado de varia enhavo en HTML-paĝojn
  - AJAX: Asynchronous Javascript and XML
  - COMET: Kunlabora, nesamtempa uz-surfaco: Demo vidu http://demo.liftweb.net/chat
  - Objekt-rilato-bildigilo | Java Persistence API
  - CRUDify-trajto (generado de Create-Read-Update-Delete-paĝoj)
  - Uzeblaj antaŭdifinoj, alte konfigurebla.
- Statike tipsekura respondo al *Ruby on Rails*

Konkludo Scala

- Potencaj, orte kombineblaj konceptoj
- Java-integrita
- Rapide kreskanta komunumo
- Facila transiro de Java
- Programilo(“IDE”)-subteno: Klaraj progresoj
- Tre taŭga kiel eduklingvo (pro sia reguleco)
- Por ampleksa aplik-logiko aŭ paraleleco
  pli komforta ol Java, pli sekura ol Ruby

Terminodiskuto

- Kontenta pri tradukoj:
  - `n = n + 1;` // no **ricevas** no plus unu
  - Singleton class: unuopul-klaso
  - Exception handling: Esceptotraktado
  - To **delegate** sth to someone: **Taskigi** iun pri io
- Duboj pri tradukoj:
  - **Assign** a value to a variable: **Stori** valoron en variablo
  - Pattern Matching: Struktura komparo (kaj elprenado)
  - Control structures: Stirordonoj
  - Language construct: lingva konstruaĵo
  - @Annotation: @Prinoto
  - Actor (parallel programming): agilo, aganto, aktoro
  - Default: Antaŭdifino