Scala Quick Reference

blablabla

Expressions

Arithmetiskt uttryck	(x + 2) * i / 3	skrivs som i matematiken, för heltal är / heltalsdivision, % ”rest”

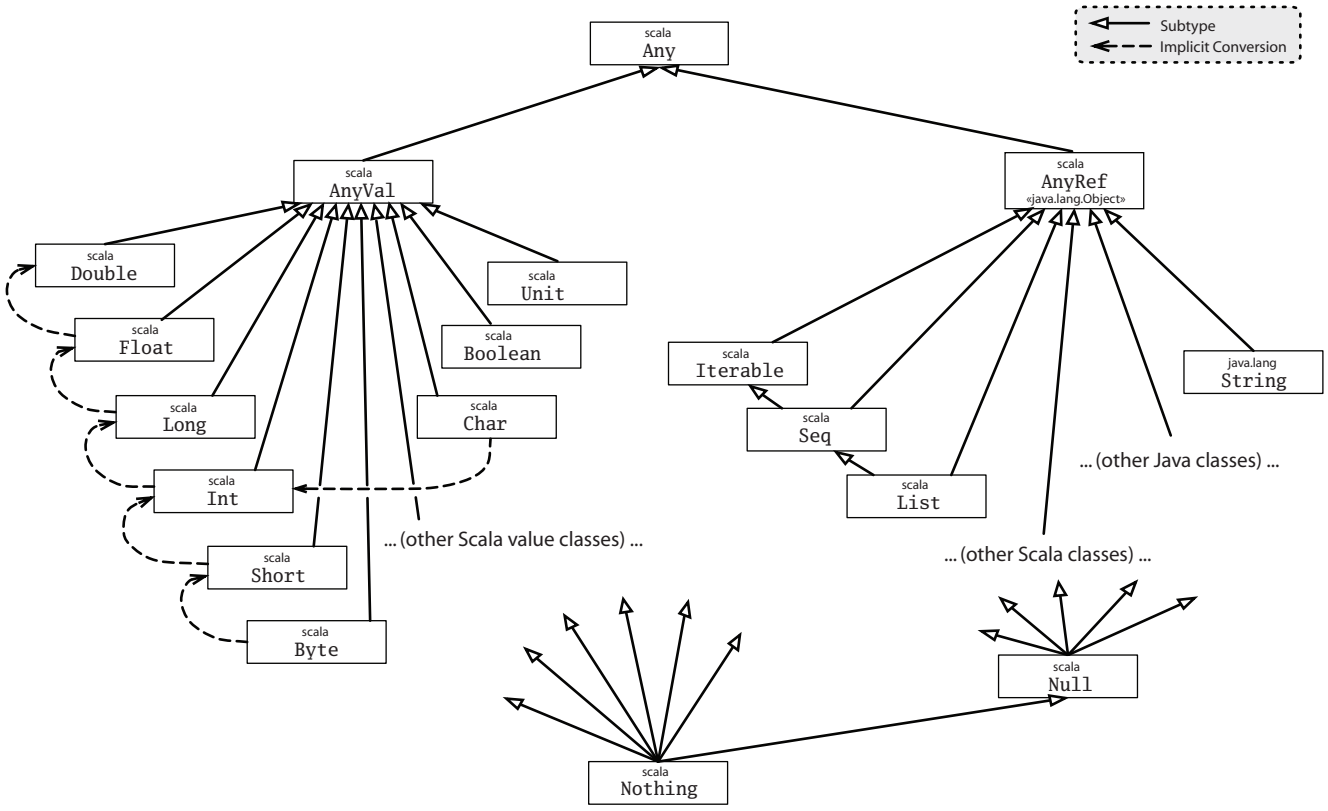
Hello `if if`

Control structures

Hello `if if`

Type system

Scala's type hierarchy



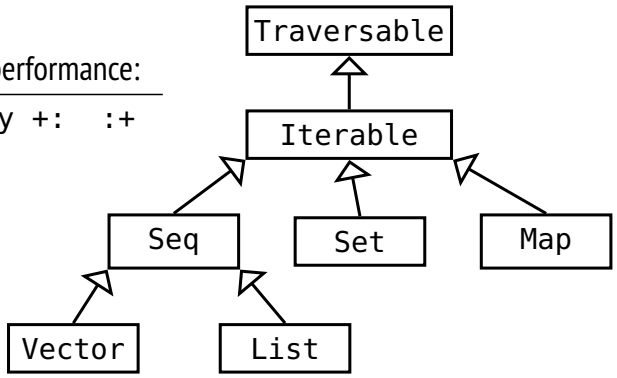
Size of basic types

name	# bits	range	JVM
Byte	8	$-2^7 \ldots 2^7 - 1$	byte
Short	16	$-2^{15} \ldots 2^{15} - 1$	short
Char	16	$0 \ldots 2^{16} - 1$	char
Int	32	$-2^{15} \ldots 2^{15} - 1$	int
Long	64	$-2^{15} \ldots 2^{15} - 1$	long
Float	32	$\pm\, 3.4028235 \cdot 10^{38}$	float
Double	64	$\pm\, 1.7976931348623157 \cdot 10^{308}$	double

Collections

scala.collection. immutable.	mutable.	methods with good performance:
Vector	ArrayBuffer	head tail apply +: :+
List	ListBuffer	head +:
Set	Set	contains + -
Map	Map	apply + -

String and Array are implicit subtypes of Seq making sequence methods work as for other collections.



Operations in trait `Traversable`

What	Example	Explanation `f` is function, `pf` is partial funct., `p` is predicate.
Traverse:	`xs foreach f`	Executes f for every element of xs. Returntype `Unit`.
Add:	`xs ++ ys`	A collection with xs followed by ys.
Map:	`xs map f`	A collection formed by applying f to every element in xs.
	`xs flatMap f`	A collection obtained by applying f (which must return a collection) to all elements in xs and concatenating the results.
	`xs collect pf`	The collection obtained by applying the pf to every element in xs for which it is defined (undefined ignored).
Convert:	`toVector toList toSeq toBuffer toArray`	Converts a collection. Unchanged if the run-time type already matches the demanded type.
	`toSet`	Converts the collection to a set; duplicates removed.
	`toMap`	Converts a collection of key/value pairs to a map.
Copy:	`xs copyToBuffer buf`	Copies all elements of xs to buffer buf. Returntype `Unit`
	`xs copyToArray (arr, s, n)`	Copies at most n elements of the collection to array arr starting at index s (last two arguments are optional). Returntype `Unit`
Size info:	`xs.isEmpty`	Returns true if the collection xs is empty.
	`xs.nonEmpty`	Returns true if the collection xs has at least one element.
	`xs.size`	Returns an `Int` with the number of elements in xs.
Retrieval:	`xs.head` `xs.last`	The first/last element of xs (or, some element, if no order is defined).
	`xs.headOption` `xs.lastOption`	The first/last element of xs (or, some element, if no order is defined) in an option value, or `None` if xs is empty.
	`xs find p`	An option with the first element satisfying p, or None.
Subparts:	`xs.tail xs.init`	The rest of the collection except xs.head or xs.last.
	`xs slice (from, to)`	The elements in from index `from` until (not including) `to`.
	`xs take n`	The first n elements (or some n elements, if order undefined).
	`xs drop n`	The rest of the collection except xs take n.
	`xs takeWhile p`	The longest prefix of elements all satisfying p.
	`xs dropWhile p`	Without the longest prefix of elements that all satisfy p.
	`xs filter p`	Those elements of xs that satisfy the predicate p.
	`xs filterNot p`	Those elements of xs that do not satisfy the predicate p.
	`xs splitAt n`	Split xs at n returning the pair (xs take n, xs drop n).
	`xs span p`	Split xs by p into the pair (xs takeWhile p, xs.dropWhile p).
	`xs partition p`	Split xs by p into the pair (xs filter p, xs.filterNot p)
	`xs groupBy f`	Partition xs into a map of collections according to f.
Conditions:	`xs forall p`	xxx
	`xs exists p`	xxx
	`xs count p`	xxx
Folds:	`xs.foldLeft(z)(op)` `xs.foldRight(z)(op)`	Apply binary operation op between successive elements of xs, going left to right (or right to left) starting with z.
	`xs reduceLeft op` `xs reduceRight op`	Same as foldLeft/foldRight, but xs must be non-empty, starting with first element instead of z.
	`xs.sum xs.product` `xs.min xs.max`	Calculation of the sum/product/min/max of the elements of xs, which must be numeric.
Make string:	`xs mkString (start, sep, end)`	A string with all elements of xs between separators sep enclosed in strings start and end; start, sep, end are all optional.

Operations in trait `Iterable`

What	Example	Explanation `f` is function, `pf` is partial funct., `p` is predicate.
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx

Operations in trait `Seq`

What	Example	Explanation `f` is function, `pf` is partial funct., `p` is predicate.
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx

Operations in trait `Set`

What	Example	Explanation `f` is function, `pf` is partial funct., `p` is predicate.
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
XXX:	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx
	xxx	xxx

Reserved words

The 40 words and 10 symbols below have special meaning and cannot be used as identifiers in Scala.

```
abstract case catch class def do else extends false final
finally for forSome if implicit import lazy macro match new
null object override package private protected return sealed
```

```
super this throw trait try true type val var while with yield
_ : = => <- <: <% >: # @
```

Java snabbreferens

Tecknet | står för ”eller”. Vanliga parenteser () används för att gruppera alternativ. Med [] markeras sådant som inte alltid finns med. Med `stmt` avses en sats, `x`, `i`, `s`, `ch` är variabler, `expr` är ett uttryck, `cond` är ett logiskt uttryck.

Satser

Block	`{stmt1; stmt2; ...}`	fungerar ”utifrån” som **en** sats
Tilldelningssats	`x = expr;`	variabeln och uttrycket av kompatibel typ
Förkortade	`x += expr;` `x++;`	x = x + expr; även −=, *=, /= x = x + 1; även x − −
if-sats	`if (cond) {stmt; ...}` `[else { stmt; ...}]`	utförs om cond är true utförs om false

switch-sats	`switch (expr) {`
	`  case A: stmt1; break;`
	`  ...`
	`  default: stmtN; break;`
	`}`
		expr är ett heltalsuttryck
		utförs om expr = A (A konstant)
		
		utförs om inget case passar

for-sats	`for (int i = start; i < stop; i++) {`
	`  stmt;`
	`  ...;`
	`}`
		
		satserna utförs för i = start, start+1, …, stop−1
		(ingen gång om start >= stop)
		i++ kan ersättas med i = i + step

while-sats	`while (cond) {`
	`  stmt; ...`
	`}`
		
		utförs så länge cond är true

do-while-sats	`do {`
	`  stmt; ...`
	`} while (cond);`
		
		utförs minst en gång,
		så länge cond är true

return-sats	`return expr;`	returnerar funktionsresultat

Uttryck

Aritmetiskt uttryck	(x + 2) * i / 3	skrivs som i matematiken, för heltal är / heltalsdivision, % ”rest”
Objektuttryck	new Classname(…) | ref-var | null | function-call | this | super	
Logiskt uttryck	! log-expr | log-expr && log-expr | log-expr || log-expr | function-call | relation | log-var | true | false	
Relation	expr (< | <= | == | >= | > | !=) expr (för objektuttryck bara == och !=, också expr instanceof Classname)	
Funktionsanrop	obj-expr.method(…) Classname.method(…)	anropa ”vanlig metod” (utför operation) anropa statisk metod
Vektor (array)	new int[size] vname[i] vname.length	skapar int-vektor med size element elementet med index i, 0..length−1 antalet element
Typkonvertering	(newtype) expr (int) real-expr (Square) aShape	konverterar expr till typen newtype – avkortar genom att stryka decimaler – ger ClassCastException om aShape inte är ett Square-objekt

Deklarationer

Allmänt	[<protection>] [static] [final] <type> name1, name2, …;	
<type>	byte | short | int | long | float | double | boolean | char | Classname	
<protection>	public | private | protected	för attribut och metoder i klasser (paketskydd om inget anges)
Startvärde	int x = 5;	startvärde bör alltid anges
Konstant	final int N = 20;	konstantnamn med stora bokstäver
Vektor	<type>[ ] vname = new <type>[10];	deklarerar och skapar vektor

Klasser

Deklaration	[public][abstract] class Classname
	[extends Classname1] [implements Interface1, Interface2, …] {
	<deklaration av attribut>
	<deklaration av konstruktorer>
	<deklaration av metoder>
	}	

Attribut	Som vanliga deklarationer. Attribut får implicita startvärden, 0, 0.0, false, null.	

Konstruktor	<prot> Classname(param, …) {
	stmt; …
	}
		Parametrarna är de parametrar som ges vid new Classname(…). Satserna ska ge attributen startvärden

Metod	<prot> <type> name(param, …) {
	stmt; …
	}
		om typen inte är void måste en return-sats exekveras i metoden

Huvudprogram	public static void main(String[ ] args) { …}	
Abstrakt metod	Som vanlig metod, men abstract före typnamnet och {. . .} ersätts med semikolon. Metoden måste implementeras i subklasserna.	

Standardklasser, java.lang, behöver inte importeras

Object	Superklass till alla klasser.	
	boolean equals(Object other);	ger true om objektet är lika med other
	int hashCode();	ger objektets hashkod
	String toString();	ger en läsbar representation av objektet
Math	Statiska konstanter Math.PI och Math.E. Metoderna är statiska (anropas med t ex Math.round(x)):	
	long round(double x);	avrundning, även float → int
	int abs(int x);	$\|x\|$, även double, …
	double hypot(double x, double y);	$\sqrt{x^2 + y^2}$
	double sin(double x);	$\sin x$, liknande: cos, tan, asin, acos, atan
	double exp(double x);	e^x
	double pow(double x, double y);	x^y
	double log(double x);	$\ln x$
	double sqrt(double x);	$\sqrt{x}$
	double toRadians(double deg);	$deg \cdot \pi/180$
System	void System.out.print(String s);	skriv ut strängen s
	void System.out.println(String s);	som print men avsluta med ny rad
	void System.exit(int status);	avsluta exekveringen, status != 0 om fel
	Parametern till print och println kan vara av godtycklig typ: int, double, …	

Typklasser

Till varje datatyp finns en typklass: char → Character, int → Integer, double → Double, … Statiska konstanter MIN_VALUE och MAX_VALUE ger minsta respektive största värde. Exempel med klassen Integer:

Integer(int value);	skapar ett objekt som innehåller value
int intValue();	tar reda på värdet

String

Teckensträngar där tecknen inte kan ändras. ”asdf” är ett String-objekt. s1 + s2 för att konkatenera två strängar. StringIndexOutOfBoundsException om någon position är fel.

int length();	antalet tecken
char charAt(int i);	tecknet på plats i, 0..length()−1
boolean equals(String s);	jämför innehållet (s1 == s2 fungerar inte)
int compareTo(String s);	< 0 om mindre, = 0 om lika, > 0 om större
int indexOf(char ch);	index för ch, −1 om inte finns
int indexOf(char ch, int from);	som indexOf men börjar leta på plats from
String substring(int first, int last);	kopia av tecknen first..last−1
String[ ] split(String delim);	ger vektor med ”ord” (ord är följder av tecken åtskilda med tecknen i delim)

Konvertering mellan standardtyp och String (exempel med int, liknande för andra typer):

String.valueOf(int x);	x = 1234 → ”1234”
Integer.parseInt(String s);	s = ”1234” → 1234, NumberFormat-Exception om s innehåller felaktiga tecken

StringBuilder

Modifierbara teckensträngar. length och charAt som String, plus:

StringBuilder(String s);	StringBuilder med samma innehåll som s
void setCharAt(int i, char ch);	ändrar tecknet på plats i till ch
StringBuilder append(String s);	lägger till s, även andra typer: int, char, …
StringBuilder insert(int i, String s);	lägger in s med början på plats i
StringBuilder deleteCharAt(int i);	tar bort tecknet på plats i
String toString();	skapar kopia som String-objekt

Standardklasser, import java.util.Classname

List

List<E> är ett gränssnitt som beskriver listor med objekt av parameterklassen E. Man kan lägga in värden av standardtyperna genom att kapsla in dem, till exempel int i Integer-objekt. Gränssnittet implementeras av klasserna ArrayList<E> och LinkedList<E>, som har samma operationer. Man ska inte använda operationerna som har en position som parameter på en LinkedList (i stället en iterator). IndexOutOfBoundsException om någon position är fel.

För att operationerna contains, indexOf och remove(Object) ska fungera måste klassen E överskugga funktionen equals(Object). Integer och de andra typklasserna gör det.

ArrayList
LinkedList

ArrayList<E>();	skapar tom lista
LinkedList<E>();	skapar tom lista
int size();	antalet element
boolean isEmpty();	ger true om listan är tom
E get(int i);	tar reda på elementet på plats i
int indexOf(Object obj);	index för obj, −1 om inte finns
boolean contains(Object obj);	ger true om obj finns i listan
void add(E obj);	lägger in obj sist, efter existerande element
void add(int i, E obj);	lägger in obj på plats i (efterföljande element flyttas)

… forts nästa sida

E set(int i, E obj);	ersätter elementet på plats i med obj
E remove(int i);	tar bort elementet på plats i (efterföljande element flyttas)

	boolean remove(Object obj);	tar bort objektet obj, om det finns
	void clear();	tar bort alla element i listan
Random	Random();	skapar ”slumpmässig” slumptalsgenerator
	Random(long seed);	– med bestämt slumptalsfrö
	int nextInt(int n);	heltal i intervallet [0, n)
	double nextDouble();	double-tal i intervallet [0.0, 1.0)
Scanner	Scanner(File f);	läser från filen f, ofta System.in
	Scanner(String s);	läser från strängen s
	String next();	läser nästa sträng fram till whitespace
	boolean hasNext();	ger true om det finns mer att läsa
	int nextInt();	nästa heltal; också nextDouble(), …
	boolean hasNextInt();	också hasNextDouble(), …
	String nextLine();	läser resten av raden

Filer, import java.io.File/FileNotFoundException/PrintWriter

Läsa från fil	Skapa en Scanner med new Scanner(new File(filename)). Ger FileNotFoundException om filen inte finns. Sedan läser man ”som vanligt” från scannern (nextInt och liknande).
Skriva till fil	Skapa en PrintWriter med new PrintWriter(new File(filename)). Ger FileNotFoundException om filen inte kan skapas. Sedan skriver man ”som vanligt” på PrintWriter-objektet (println och liknande).
Fånga undantag	Så här gör man för att fånga FileNotFoundException:

```
Scanner scan = null;
try {
    scan = new Scanner(new File("indata.txt"));
} catch (FileNotFoundException e) {
    … ta hand om felet
}
```

Specialtecken

Några tecken måste skrivas på ett speciellt sätt när de används i teckenkonstanter:

\n	ny rad, radframmatningstecken
\t	ny kolumn, tabulatortecken (eng. tab)
\\	bakåtsnedstreck: \ (eng. backslash)
\”	citationstecken: ”
\’	apostrof: ’

Reserverade ord

Nedan 50 ord kan ej användas som identifierare i Java. Orden **goto** och **const** är reserverade men används ej.

```
abstract assert boolean break byte case catch char class const
continue default do double else enum extends final finally float for
goto if implements import instanceof int interface long native new
package private protected public return short static strictfp super
switch synchronized this throw throws transient try void volatile while
```