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

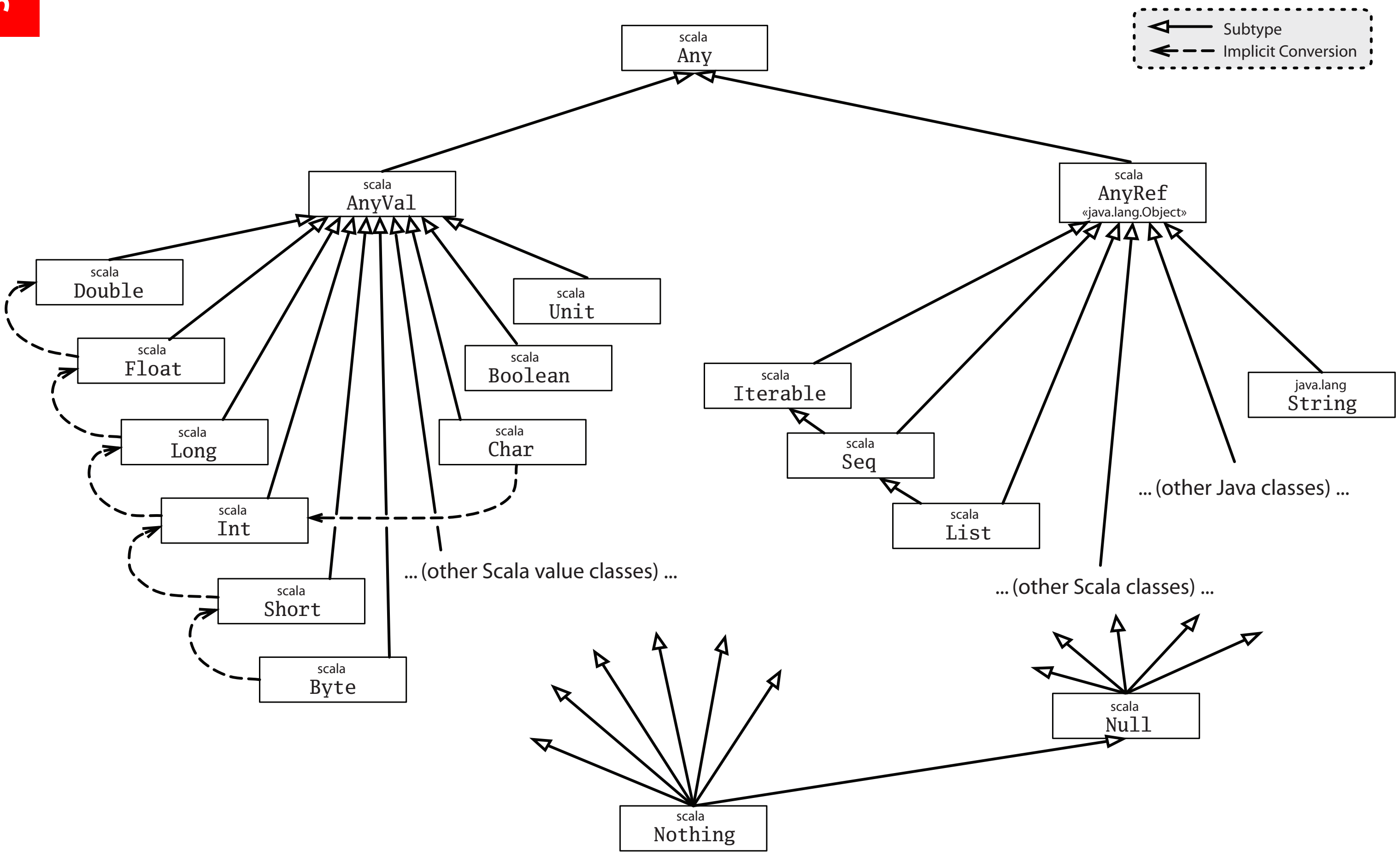
Hello `if` `if`

Option, Some, None

scala.util.Try

scala.concurrent.Future

The Scala Type System



Basic types

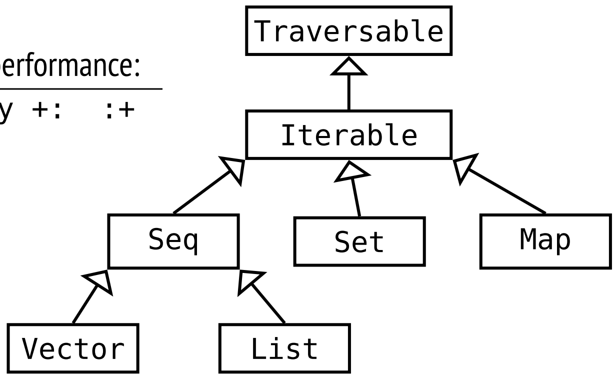
name	# bits	range	Litteral	JVM
`Byte`	8	$-2^7 \ldots 2^7-1$	`0.toByte`	`byte`
`Short`	16	$-2^{15} \ldots 2^{15}-1$	`0.toShort`	`short`
`Char`	16	$0 \ldots 2^{16}-1$	`'0'`	`char`
`Int`	32	$-2^{15} \ldots 2^{15}-1$	`0`	`int`
`Long`	64	$-2^{15} \ldots 2^{15}-1$	`0L`	`long`
`Float`	32	$\pm 3.4028235 \cdot 10^{38}$	`0F`	`float`
`Double`	64	$\pm 1.7976931348623157 \cdot 10^{308}$	`0.0`	`double`

The Scala Standard Collection Library

`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 implicitly converted to `Seq` making sequence methods work as for other collections.
Allocate Int array of size n: `new Array[Int](n)`



Concrete implementations of Set include HashSet, ListSet and BitSet. The subtype SortedSet is implemented by TreeSet. Concrete implementations of Map include HashMap and ListMap. The subtype SortedMap is implemented by TreeMap.

Methods in trait `Traversable`

What	**Usage**	**Explanation** f is a function, pf is a partial funct., p is a predicate.
Traverse:	`xs foreach f`	Executes f for every element of xs. Return type 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. Return type 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). Return type 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 elem, if order undefined).
	`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`	Returns true if p holds for all elements of xs.
	`xs exists p`	Returns true if p holds for some element of xs.
	`xs count p`	An `Int` with the number of elements in xs that satisfy p.

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`
		Similar to 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.

Methods in trait `Iterable`

What	Usage	Explanation
Iterators:	`val it = xs.iterator`	An iterator `it` of type `Iterator` that yields each element one by one: `while (it.hasNext) f(it.next)`
	`xs grouped size`	An iterator yielding fixed-sized chunks of this collection.
	`xs sliding size`	An iterator yielding a sliding fixed-sized window of elements.

Subparts:	`xs takeRight n`
	`xs dropRight n`
		Similar to `take` and `drop` in `Traversable` but takes/drops the last n elements (or any n elements if the order is undefined).

Zippers:	`xs zip ys`	An iterable of pairs of corresponding elements from xs and ys.
	`xs zipAll (ys, x, y)`	Similar to `zip`, but the shorter sequence is extended to match the longer one by appending elements x or y.
	`xs.zipWithIndex`	An iterable of pairs of elements from xs with their indices.
Compare:	`xs sameElements ys`	True if xs and ys contain the same elements in the same order.

Methods in trait Seq

Indexing and size:	`xs(i)` `xs apply i`	The element of xs at index i.
	`xs.length`	Length of sequence. Same as `size` in `Traversable`.
	`xs.indices`	Returns a Range extending from 0 to xs.length - 1.
	`xs isDefinedAt i`	True if i is contained in xs.indices.
	`xs lengthCompare n`	Returns -1 if xs is shorter than n, +1 if it is longer, else 0.
Index search:	`xs indexOf x`	The index of the first element in xs equal to x.
	`xs lastIndexOf x`	The index of the last element in xs equal to x.

	`xs indexOfSlice ys`
	`xs lastIndexOfSlice ys`
		The (last) index of xs such that successive elements starting from that index form the sequence ys.

	`xs indexWhere p`	The index of the first element in xs that satisfies p.
	`xs segmentLength (p, i)`	The length of the longest uninterrupted segment of elements in xs, starting with xs(i), that all satisfy the predicate p.
	`xs prefixLength p`	Same as `xs.segmentLength(p, 0)`
Add:	`x +: xs` `xs :+ x`	Prepend/Append x to xs. Colon on the collection side.
	`xs padTo (len, x)`	Append the value x to xs until length len is reached.
Update:	`xs patch (i, ys, r)`	A copy of xs with r elements of xs replaced by ys starting at i.
	`xs updated (i, x)`	A copy of xs with the element at index i replaced by x.

	`xs(i) = x`
	`xs.update(i, x)`
		Only available for mutable sequences. Changes the element of xs at index i to x. Return type Unit.

Sort:	`xs.sorted`	A new Seq[A] sorted using implicitly available ordering of A.
	`xs sortWith lt`	A new Seq[A] sorted using less than lt: (A, A) => Boolean.
	`xs sortBy f`	A new Seq[A] sorted using implicitly available ordering of B after applying f: A => B to each element.
Reverse:	`xs.reverse`	A new sequence with the elements of xs in reverse order.
	`xs.reverseIterator`	An iterator yielding all the elements of xs in reverse order.
	`xs reverseMap f`	Similar to map in Traversable, but in reverse order.
Tests:	`xs startsWith ys`	True if xs starts with sequence ys.
	`xs endsWith ys`	True if xs ends with sequence ys.
	`xs contains x`	True if xs has an element equal to x.
	`xs containsSlice ys`	True if xs has a contiguous subsequence equal to ys
	`(xs corresponds ys)(p)`	True if corresponding elements satisfy the binary predicate p.
Subparts:	`xs intersect ys`	The intersection of xs and ys, preserving element order.
	`xs diff ys`	The difference of xs and ys, preserving element order.
	`xs union ys`	Same as `xs ++ ys` in Traversable.
	`xs.distinct`	A subsequence of xs that contains no duplicated element.

Methods in trait `Set`

`xs(x)   xs apply x`	True if x is a member of xs. Also: xs contains x
`xs subsetOf ys`	True if ys is a subset of xs.

`xs + x          xs - x`
`xs + (x, y, z)  xs - (x, y, z)`
	Returns a new set including/excluding elements.
	Addition/subtraction can be applied to many arguments.

`xs intersect ys`	A new set with elements in both xs and ys. Also: &
`xs union ys`	A new set with elements in either xs or ys or both. Also: |
`xs diff ys`	A new set with elements in xs that are not in ys. Also: &~

Additional methods only in trait `mutable.Set`

`xs += x          xs -= x`
`xs += (x, y, z)  xs -= (x, y, z)`
	Returns the same set with included/excluded elements.
	Addition/subtraction can be applied to many arguments.

`xs ++= ys`	Adds all elements in ys to set xs and returns xs itself.
`xs add x`	Adds element x to xs and returns true if x was in xs, else false.
`xs remove x`	Removes x from xs and returns true if x was in xs, else false.
`xs retain p`	Keeps only those elements in xs that satisfy predicate p.
`xs.clear`	Removes all elements from xs. Return type Unit.
`xs(x) = b     xs.update(x, b)`	If b is true, adds x to xs, else removes x. Return type Unit.
`xs.clone`	Returns a new mutable set with the same elements as xs.

Methods in trait `Map`

`ms get k`	The value associated with key k an option, None if not found.
`ms(k)   xs apply k`	The value associated with key k, or exception if not found.
`ms getOrElse (k, d)`	The value associated with key k in map ms, or d if not found.
`ms isDefinedAt k`	True if ms contains a mapping for key k. Also: ms.contains(k)

`ms + (k -> v)     ms + ((k, v))`
`ms updated (k, v)`
	The map containing all mappings of ms as well as the mapping
	k -> v from key k to value v. Also: ms + (k -> v, l -> w)

`ms - k`	Excluding any mapping of key k. Also: ms - (k, l, m)
`ms ++ ks       ms -- ks`	The mappings of ms with the mappings of ks added/removed.
`ms.keys        ms.values`	An iterable containing each key/value in ms.

Additional methods only in trait `mutable.Map`

`ms(k) = v     ms.update(k, v)`	Adds mapping k to v, overwriting any previous mapping of k.
`ms += (k -> v)       ms -= k`	Adds/Removes mappings. Also vid several arguments.
`ms put (k, v)      ms remove k`	Adds/removes mapping; returns previous value of k as an option.
`ms retain p`	Keeps only mappings that have a key satisfying predicate p.
`ms.clear`	Removes all mappings from ms.
`ms transform f`	Transforms all associated values in map ms with function f.
`ms.clone`	Returns a new mutable map with the same mappings as ms.

Factory methods examples: `Vector(1, 2, 3); collection.mutable.Set.empty[Int]; Map("Sweden" -> "Stockholm", "Denmark" -> "Copenhagen"); List.fill(3)('a');` `Array.ofDim[Int](3,2)` gives `Array(Array(0, 0), Array(0, 0), Array(0, 0))` same as `Array.fill(3,2)(0); Vector.iterate(1.2, 3)(_ + 0.5)` gives `Vector(1.2, 1.7, 2.2); Vector.tabulate(3)("s" + _)` gives `Vector("s0", "s1", "s2")`

String methods

Some methods below are from java.lang.String and some methods are implicitly added from StringOps, etc.
Strings are implictly treated as Seq[Char] so all Seq methods also works.

`s.capitalize`	Returns this string with first character converted to upper case.
`s(i)  s apply i   s.charAt(i)`	Returns the character at index i.
`s.compareTo(t)`	Returns x where x < 0 if s < t, x > 0 if s > t, x is 0 if s == t
`s.compareToIgnoreCase(t)`	Similar to compateTo but not sensitive to case.
`s.endsWith(t)`	True if string s ends with string t.
`s.replaceAllLiterally(s1, s2)`	Replace all occurances of s1 with s2 in s.
`s.split(c)`	Returns an array of strings split at every occurance of charachter c.
`s.startsWith(t)`	True if string s begins with string t.
`s.stripMargin`	Strips leading white space followed by | from each line in string.
`s.substring(i)`	Returns a substring of s with all charcters from index i.
`s.substring(i, j)`	Returns a substring of s from index i to index j-1.
`s.toInt  s.toDouble  s.toFloat`	Parses s as an Int or Double etc. May throw an exception.
`42.toString   42.0.toString`	Converts a number to a String.
`s.toLowerCase`	Converts all characters to lower case.
`s.toUpperCase`	Converts all characters to upper case.
`s.trim`	Removes leading and trailing white space.

scala.io.Source

scala.io.StdIn

Special characters and strings

Escape char		**String**	
`\n`	line break	`"hello\nworld"`	string including escape char for line break
`\t`	horisontal tab	`"""a "raw" string"""`	can include quotes and span multiple lines
`\"`	double quote "	`s"x is $x"`	the s interpolator inserts values of existing names
`\'`	single quote '	`s"x+1 is ${x+1}"`	the s interpolator evaluates expressions within `${}`
`\\`	backslash \		

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 $\longrightarrow$ int
	int abs(int x);	$\lvert x \rvert$, 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, NumberFormatException 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	ArrayList<E>();	skapar tom lista
LinkedList	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
```