

# Scala Quick Reference

blablabla

## Control structures

---

Hello **if if**

# Control structures

---

Hello **if if**

Hello **if if**

Hello

# 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	<code>{stmt1; stmt2; ...}</code>	fungerar "utifrån" som <b>en</b> sats
Tilldelningssats	<code>x = expr;</code>	variabeln och uttrycket av kompatibel typ
Förkortade	<code>x += expr;</code> <code>x++;</code>	<code>x = x + expr</code> ; även <code>--</code> , <code>*=</code> , <code>/=</code> <code>x = x + 1</code> ; även <code>x --</code>
if-sats	<b>if</b> ( <code>cond</code> ) { <code>stmt</code> ; ...} <b>[else</b> { <code>stmt</code> ; ...} <b>]</b>	utförs om <code>cond</code> är true utförs om false
switch-sats	<b>switch</b> ( <code>expr</code> ) { <b>case</b> A: <code>stmt1</code> ; <b>break</b> ; ... <b>default</b> : <code>stmtN</code> ; <b>break</b> ; }	<code>expr</code> är ett heltalsuttryck utförs om <code>expr = A</code> (A konstant) utförs om inget case passar
for-sats	<b>for</b> ( <b>int</b> <code>i</code> = <code>start</code> ; <code>i</code> < <code>stop</code> ; <code>i++</code> ) { <code>stmt</code> ; ...; }	satserna utförs för <code>i = start, start+1, ..., stop-1</code> (ingen gång om <code>start &gt;= stop</code> ) <code>i++</code> kan ersättas med <code>i = i + step</code>
while-sats	<b>while</b> ( <code>cond</code> ) { <code>stmt</code> ; ... }	utförs så länge <code>cond</code> är true
do-while-sats	<b>do</b> { <code>stmt</code> ; ... } <b>while</b> ( <code>cond</code> );	utförs minst en gång, så länge <code>cond</code> är true
return-sats	<b>return</b> <code>expr</code> ;	returnerar funktionsresultat

## Uttryck

Aritmetiskt uttryck	<code>(x + 2) * i / 3</code>	skrivs som i matematiken, för heltal är / heltalsdivision, % "rest"
Objektuttryck	<code>new Classname(...)</code>   <code>ref-var</code>   <code>null</code>   <code>function-call</code>   <code>this</code>   <code>super</code>	
Logiskt uttryck	<code>! log-expr</code>   <code>log-expr &amp;&amp; log-expr</code>   <code>log-expr    log-expr</code>   <code>function-call</code>   <code>relation</code>   <code>log-var</code>   <code>true</code>   <code>false</code>	
Relation	<code>expr ( &lt;   &lt;=   ==   &gt;=   &gt;   != ) expr</code> (för objektuttryck bara <code>==</code> och <code>!=</code> , också <code>expr instanceof Classname</code> )	
Funktionsanrop	<code>obj-expr.method(...)</code> <code>Classname.method(...)</code>	anropa "vanlig metod" (utför operation) anropa statisk metod
Vektor (array)	<code>new int[size]</code> <code>vname[i]</code> <code>vname.length</code>	skapar int-vektor med <code>size</code> element elementet med index <code>i</code> , <code>0..length-1</code> antalet element
Typkonvertering	<code>(newtype) expr</code> <code>(int) real-expr</code> <code>(Square) aShape</code>	konverterar <code>expr</code> till typen <code>newtype</code> – avkortar genom att stryka decimaler – ger <code>ClassCastException</code> om <code>aShape</code> inte är ett <code>Square</code> -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 konstruktörer> <deklaration av metoder> }	
Attribut	Som vanliga deklarerationer. Attribut får implicita startvärden, 0, 0.0, false, null.	
Konstruktör	<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 subclasserna.	

### Standardklasser, java.lang, behöver inte importeras

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

Typklasser	Till varje datatyp finns en typklass: <code>char</code> → <code>Character</code> , <code>int</code> → <code>Integer</code> , <code>double</code> → <code>Double</code> , ... Statiska konstanter <code>MIN_VALUE</code> och <code>MAX_VALUE</code> ger minsta respektive största värde. Exempel med klassen <code>Integer</code> :	
	<code>Integer(int value);</code> <code>int intValue();</code>	skapar ett objekt som innehåller value tar reda på värdet
String	Teckensträngar där tecknen inte kan ändras. "asdf" är ett String-objekt. <code>s1 + s2</code> för att konkatenera två strängar. <code>StringIndexOutOfBoundsException</code> om någon position är fel.	
	<code>int length();</code> <code>char charAt(int i);</code> <code>boolean equals(String s);</code> <code>int compareTo(String s);</code> <code>int indexOf(char ch);</code> <code>int indexOf(char ch, int from);</code> <code>String substring(int first, int last);</code> <code>String[] split(String delim);</code>	antalet tecken tecknet på plats i, <code>0..length()-1</code> jämför innehållet ( <code>s1 == s2</code> fungerar inte) < 0 om mindre, = 0 om lika, > 0 om större index för ch, -1 om inte finns som <code>indexOf</code> men börjar leta på plats from kopia av tecknen <code>first..last-1</code> 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):	
	<code>String.valueOf(int x);</code> <code>Integer.parseInt(String s);</code>	<code>x = 1234</code> → "1234" <code>s = "1234"</code> → 1234, <code>NumberFormatException</code> om s innehåller felaktiga tecken
StringBuilder	Modifierbara teckensträngar. <code>length</code> och <code>charAt</code> som String, plus:	
	<code>StringBuilder(String s);</code> <code>void setCharAt(int i, char ch);</code> <code>StringBuilder append(String s);</code> <code>StringBuilder insert(int i, String s);</code> <code>StringBuilder deleteCharAt(int i);</code> <code>String toString();</code>	StringBuilder med samma innehåll som s ändrar tecknet på plats i till ch lägger till s, även andra typer: int, char, ... lägger in s med början på plats i tar bort tecknet på plats i 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 <code>ArrayList&lt;E&gt;</code> och <code>LinkedList&lt;E&gt;</code> , som har samma operationer. Man ska inte använda operationerna som har en position som parameter på en <code>LinkedList</code> (i stället en iterator). <code>IndexOutOfBoundsException</code> om någon position är fel.  För att operationerna <code>contains</code> , <code>indexOf</code> och <code>remove(Object)</code> ska fungera måste klassen E över-skugga funktionen <code>equals(Object)</code> . Integer och de andra typklasserna gör det.	
ArrayList	<code>ArrayList&lt;E&gt;();</code>	skapar tom lista
LinkedList	<code>LinkedList&lt;E&gt;();</code>	skapar tom lista
	<code>int size();</code>	antalet element
	<code>boolean isEmpty();</code>	ger true om listan är tom
	<code>E get(int i);</code>	tar reda på elementet på plats i
	<code>int indexOf(Object obj);</code>	index för obj, -1 om inte finns
	<code>boolean contains(Object obj);</code>	ger true om obj finns i listan
	<code>void add(E obj);</code>	lägger in obj sist, efter existerande element
	<code>void add(int i, E obj);</code>	lägger in obj på plats i (efterföljande element flyttas)
	... forts nästa sida	
	<code>E set(int i, E obj);</code>	ersätter elementet på plats i med obj
	<code>E remove(int i);</code>	tar bort elementet på plats i (efter-följande element flyttas)

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

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:

<code>\n</code>	radmatning
<code>\t</code>	tab
<code>\\</code>	bakåtsnedstreck ( <code>\</code> , eng. backslash)
<code>\"</code>	citationstecken ( <code>"</code> )
<code>\'</code>	apostrof ( <code>'</code> )