

# Java snabbreferens (2013–01–15)

av Per Holm, [Per.Holm@cs.lth.se](mailto:Per.Holm@cs.lth.se)

Här beskrivs delar av Java på ett kortfattat sätt. Tecknet | står för ”eller”; vanliga parenteser ( ) används för att gruppera alternativ. Med [ ] markeras sådant som inte alltid finns med. stmt = sats, var = variabel, expr = uttryck, cond = logiskt uttryck.

## Satser

---

Block	{ stmt1; stmt2; ...; }	// fungerar ”utifrån” som <i>en</i> sats
Tilldelningssats	var = expr;	// variabeln och uttrycket av kompatibel typ
Förkortade	var += expr; var++;	// var = var + expr; även -=, *=, /= // var = var + 1; även var --
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) * z / 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 konstruktörer> <deklaration av metoder> }
Attribut	Som vanliga deklarerationer. Attribut får implicita startvärden, 0, 0.0, false, null.
Konstruktör	<prot> Classname(param, ...) { // Parametrarna är de parametrar som ges vid stmt; ... // new Classname(...). Satserna ska ge } // attributen startvärden
Metod	<prot> <type> name(param, ...) { // om typen inte är void måste en return- stmt; ... // 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	<p>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:</p> <pre>Integer(int value);           // skapar ett objekt som innehåller value int intValue();              // tar reda på värdet</pre>
String	<p>Teckensträngar där tecknen inte kan ändras. "asdf" är ett String-objekt. s1 + s2 för att konkatenera två strängar. StringIndexOutOfBoundsException om någon position är fel.</p> <pre>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);    // &lt; 0 om mindre, = 0 om lika, &gt; 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 följer av                                // tecken åtskilda med tecknen i delim)</pre> <p>Konvertering mellan standardtyp och String (exempel med int, liknande för andra typer):</p> <pre>String.valueOf(int x);       // x = 1234 → "1234" Integer.parseInt(String s);  // s = "1234" → 1234, NumberFormat-                                // Exception om s innehåller felaktiga tecken</pre>
StringBuilder	<p>Modifierbara teckensträngar. length och charAt som String, plus:</p> <pre>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</pre>

## Standardklasser, import java.util.Classname

List	<p>List&lt;E&gt; 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&lt;E&gt; och LinkedList&lt;E&gt;, 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.</p> <p>För att operationerna contains, indexOf och remove(Object) ska fungera måste klassen E överskugga funktionen equals(Object). Integer och de andra typklasserna gör det.</p>
ArrayList LinkedList	<pre>ArrayList&lt;E&gt;();              // skapar tom lista LinkedList&lt;E&gt;();             // 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)</pre> <p>... forts nästa sida</p>

	E set(int i, E obj);	// ersätter elementet på plats i med obj
	E remove(int i);	// tar bort elementet på plats i (efter- // följande 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

## **Files, 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
}
```