Core Java

By Cay S. Horstmann; Revised & Updated by Ivan St. Ivanov

ABOUT CORE JAVA

This Refcard gives you an overview of key aspects of the Java language and cheat sheets on the core library (formatted output, collections, regular expressions, logging, properties) as well as the most commonly used tools (`javac`, `java`, `jar`).

JAVA KEYWORDS

KEYWORD	DESCRIPTION	EXAMPLE
abstract	an abstract class or method	`abstract class Writable { public abstract void write(Writer out); public void save(String filename) { ... } }`
assert	with assertions enabled, throws an error if condition not fulfilled	`assert param != null;` **Note:** Run with **-ea** to enable assertions
boolean	the Boolean type with values true and false	`boolean more = false;`
break	breaks out of a switch or loop	`while ((ch = in.next()) != -1) { if (ch == '\n') break; process(ch); }` **Note:** Also see **switch**

byte	the 8-bit integer type	`byte b = -1; // Not the same as 0xFF`
		Note: Be careful with bytes < 0

case	a case of a switch	see **switch**
catch	the clause of a try block catching an exception	see **try**
char	the Unicode character type	`char input = 'Q';`
class	defines a class type	`class Person { private String name; public Person(String aName) { name = aName; } public void print() { System.out.println(name); } }`
continue	continues at the end of a loop	`while ((ch = in.next()) != -1) { if (ch == ' ') continue; process(ch); }`
default	1) the default clause of a switch	see **switch**
default *(cont.)*	2) denotes default implementation of an interface method	`public interface Collection<E> { @Override default Spliterator<E> spliterator() { return Spliterators.spliterator(this, 0); } }`
do	the top of a do/while loop	`do { ch = in.next(); } while (ch == ' ');`
double	the double-precision floating-number type	`double oneHalf = 0.5;`
else	the else clause of an if statement	see **if**
enum	an enumerated type	`enum Mood { SAD, HAPPY };`
extends	defines the parent class of a class	`class Student extends Person { private int id; public Student(String name, int anId) { ... } public void print() { ... } }`
final	a constant, or a class or method that cannot be overridden	`public static final int DEFAULT_ID = 0;`
finally	the part of a try block that is always executed	see **try**
float	the single-precision floating-point type	`float oneHalf = 0.5F;`

KEYWORD	DESCRIPTION	EXAMPLE

for	a loop type	`for (int i = 10; i >= 0; i--)`
		`  System.out.println(i);`
		`for (String s : line.split("\\s+"))`
		`  System.out.println(s);`
		Note: In the "generalized" for loop, the expression after the : must be an array or an `Iterable`

if	a conditional statement	`if (input == 'Q')`
		`  System.exit(0);`
		`else`
		`  more = true;`

implements	defines the interface(s) that a class implements	`class Student`
		`implements Printable {`
		`...`
		`}`

import	imports a package	`import java.util.ArrayList;` `import com.dzone.refcardz.*;`

instanceof	tests if an object is an instance of a class	`if (fred instanceof Student)`
		`  value = ((Student) fred).getId();`
		Note: `null instanceof T` is always false

int	the 32-bit integer type	`int value = 0;`

interface	an abstract type with methods that a class can implement	`interface Printable {`
		`  void print();`
		`}`

long	the 64-bit long integer type	`long worldPopulation = 6710044745L;`
native	a method implemented by the host system	
new	allocates a new object or array	`Person fred = new Person("Fred");`
null	a null reference	`Person optional = null;`
package	a package of classes	`package com.dzone.refcardz;`
private	a feature that is accessible only by methods of this class	see **class**

protected	a feature that is accessible only by methods of this class, its children, and other classes in the same package	`class Student {`
		`  protected int id;`
		`  ...`
		`}`

public	a feature that is accessible by methods of all classes	see **class**
return	returns from a method	`int getId() { return id; }`
short	the 16-bit integer type	`short skirtLength = 24;`

static	a feature that is unique to its class, not to objects of its class	`public class WriteUtil {`
		`  public static void write(Writable[] ws,`
		`  String filename);`
		`  public static final String DEFAULT_EXT = ".dat";`
		`}`

KEYWORD	DESCRIPTION	EXAMPLE
strictfp	Use strict rules for floating-point computations	

super	invoke a superclass constructor or method	`public Student(String name, int anId) {`
		`  super(name); id = anId;`
		`}`
		`public void print() {`
		`  super.print();`
		`  System.out.println(id);`
		`}`

switch	a selection statement	`switch (ch) {`
		`  case 'Q':`
		`  case 'q':`
		`  more = false; break;`
		`  case ' ';`
		`  break;`
		`  default:`
		`  process(ch); break;`
		`}`
		Note: If you omit a break, processing continues with the next case.

synchronized	a method or code block that is atomic to a thread	`public synchronized void addGrade(String gr) {`
		`  grades.add(gr);`
		`}`

this	the implicit argument of a method, or a constructor of this class	`public Student(String id) {this.id = id;}`
		`public Student() {`
		`this(""); }`

throw	throws an exception	`if (param == null)` `  throw new IllegalArgumentException();`
throws	the exceptions that a method can throw	`public void print()` `  throws PrinterException, IOException`

transient	marks data that should not be persistent	`class Student {`
		`  private transient Data cachedData;`
		`  ...`
		`}`

try	a block of code that traps exceptions	`try {`
		`  try {`
		`  fred.print(out);`
		`  } catch (PrinterException ex) {`
		`  ex.printStackTrace();`
		`  }`
		`} finally {`
		`  out.close();`
		`}`

void	denotes a method that returns no value	`public void print() { ... }`

volatile	ensures that a field is coherently accessed by multiple threads	`class Student {`
		`  private volatile int nextId;`
		`  ...`
		`}`

while	a loop	`while (in.hasNext())` `  process(in.next());`

STANDARD JAVA PACKAGES

java.applet	Applets (Java programs that run inside a web page)
java.awt	Graphics and graphical user interfaces

java.beans	Support for JavaBeans components (classes with properties and event listeners)
java.io	Input and output
java.lang	Language support
java.math	Arbitrary-precision numbers
java.net	Networking
java.nio	"New" (memory-mapped) I/O
java.rmi	Remote method invocations
java.security	Security support
java.sql	Database support
java.text	Internationalized formatting of text and numbers
java.time	Dates, time, duration, time zones, etc.
java.util	Utilities (including data structures, concurrency, regular expressions, and logging)

OPERATOR PRECEDENCE

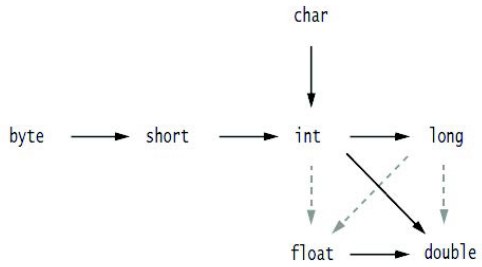
OPERATORS WITH THE SAME PRECEDENCE		NOTES
[] . () (method call)	Left to right	
! ~ ++ -- + (unary) - (unary) () (cast) new	Right to left	~ flips each bit of a number
* / %	Left to right	Be careful when using % with negative numbers. -a % b == -(a % b), but a % -b == a % b. For example, -7 % 4 == -3, 7 % -4 == 3
+ -	Left to right	
<< >> >>>	Left to right	>> is arithmetic shift (n >> 1 == n / 2 for positive and negative numbers), >>> is logical shift (adding 0 to the highest bits). The right hand side is reduced modulo 32 if the left hand side is an int or modulo 64 if the left hand side is a long. For example, 1 << 35 == 1 << 3
< <= > >= instanceof	Left to right	null instanceof T is always false
== !=	Left to right	Checks for identity. Use equals to check for structural equality
&	Left to right	Bitwise AND; no lazy evaluation with bool arguments
^	Left to right	Bitwise XOR
|	Left to right	Bitwise OR; no lazy evaluation with bool arguments
&&	Left to right	
||	Left to right	
?:	Right to left	
= += -= *= /= %= &= |= ^= <<= >>= >>>=	Right to left	

PRIMITIVE TYPES

TYPE	SIZE	RANGE	NOTES
int	4 bytes	–2,147,483,648 to 2,147,483,647 (just over 2 billion)	The wrapper type is Integer. Use BigInteger for arbitrary precision integers
short	2 bytes	–32,768 to 32,767	
long	8 bytes	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	Literals end with L (e.g. 1L)
byte	1 byte	–128 to 127	Note that the range is not 0 to 255
float	4 bytes	approximately ±3.40282347E+38F (6–7 significant decimal digits)	Literals end with F (e.g. 0.5F)
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	Use BigDecimal for arbitrary precision floating-point numbers
char	2 bytes	\u0000 to \uFFFF	The wrapper type is Character. Unicode characters > U+FFFF require two char values
boolean		true or false	

LEGAL CONVERSIONS BETWEEN PRIMITIVE TYPES

Dotted arrows denote conversions that may lose precision.



LAMBDA EXPRESSIONS

FUNCTIONAL INTERFACES

Interfaces with a single abstract method. Example:

```
@FunctionalInterface
public interface Predicate<T> {
    boolean test(T t);
}
```

Implementations of this interface can be supplied in-line as a ***lambda expression***:

- Anonymous implementations of functional interfaces
- Parameters and body are separated by an arrow sign ("->")
- Parameters of the abstract method are on the left of the arrow
- The implementation is on the right of the arrow

Typical usage of lambda expressions:

```
JButton button = new JButton("MyButton");
button.addActionListener(event ->
doSomeImportantStuff(event));
```

METHOD REFERENCES

Lambda expressions represent anonymous functions. You can pass them as method parameters or return them. The same can be done with named methods using method references.

Typical usage of method references:

WITHOUT METHOD REFERENCE	WITH METHOD REFERENCE
`button.addActionListener(event -> doSomeImportantStuff(event));`	`button.addActionListener( this::doSomeImportantStuff);`
`list.forEach(element -> System.out.println(element));`	`list.forEach(System. out::println);`

There are four kinds of method references:

KIND OF METHOD REFERENCE	EXAMPLE
To a static method	`Collections::emptyList`
To an instance method of a particular (named) object	`user::getFirstName`
To an instance method of an arbitrary object (to be named later) of a given type	`User::getFirstName`
To a constructor	`User::new`

COLLECTIONS & COMMON ALGORITHMS

`ArrayList`	An indexed sequence that grows and shrinks dynamically
`LinkedList`	An ordered sequence that allows efficient insertions and removal at any location
`ArrayDeque`	A double-ended queue that is implemented as a circular array
`HashSet`	An unordered collection that rejects duplicates
`TreeSet`	A sorted set
`EnumSet`	A set of enumerated type values
`LinkedHashSet`	A set that remembers the order in which elements were inserted
`PriorityQueue`	A collection that allows efficient removal of the smallest element
`HashMap`	A data structure that stores key/value associations
`TreeMap`	A map in which the keys are sorted
`EnumMap`	A map in which the keys belong to an enumerated type
`LinkedHashMap`	A map that remembers the order in which entries were added
`WeakHashMap`	A map with values that can be reclaimed by the garbage collector if they are not used elsewhere
`IdentityHashMap`	A map with keys that are compared by ==, not equals

COMMON TASKS

`List<String> strs = new ArrayList<>();`	Collect strings
`strs.add("Hello"); strs. add("World!");`	Add strings
`for (String str : strs) System. out.println(str);`	Do something with all elements in the collection
`Iterator<String> iter = strs. iterator(); while (iter.hasNext()) { String str = iter.next(); if (someCondition(str)) iter. remove(); }`	Remove elements that match a condition. The `remove` method removes the element returned by the preceding call to `next`
`strs.addAll(strColl);`	Add all strings from another collection of strings
`strs.addAll(Arrays.asList(args))`	Add all strings from an array of strings. `Arrays.asList` makes a `List` wrapper for an array
`strs.removeAll(coll);`	Remove all elements of another collection. Uses equals for comparison
`if (0 <= i && i < strs.size()) { str = strs.get(i); strs.set(i, "Hello"); }`	Get or set an element at a specified index
`strs.insert(i, "Hello"); str = strs.remove(i);`	Insert or remove an element at a specified index, shifting the elements with higher index values
`String[] arr = new String[strs. size()]; strs.toArray(arr);`	Convert from collection to array
`String[] arr = ...; List<String> lst = Arrays. asList(arr); lst = Arrays.asList("foo", "bar", "baz");`	Convert from array to list. Use the varargs form to make a small collection
`List<String> lst = ...; lst.sort(); lst.sort((a, b) -> a.length() - b.length());`	Sort a list by the natural order of the elements, or with a custom comparator
`Map<String, Person> map = new LinkedHashMap<String, Person>();`	Make a map that is traversed in insertion order (requires `hashCode` for key type). Use a `TreeMap` to traverse in sort order (requires that key type is comparable)
`for (Map.Entry<String, Person> entry : map.entrySet()) { String key = entry.getKey(); Person value = entry. getValue(); ... }`	Iterate through all entries of the map
`Person key = map.get(str); // null if not found map.put(key, value);`	Get or set a value for a given key

BULK OPERATIONS WITH STREAM API

`strs.forEach(System.out::println);`	Do something with all elements in the collection
`List<String> filteredList = strs .stream() .filter(this::someCondition) .collect(Collectors.toList());`	Filter elements that match a condition

`String concat = strs .stream() .collect(Collectors.joining(", "));`	Concatenate the elements of a stream
`List<User> users = ...; List<String> firstNames = users .stream() .map(User::getFirstName) .collect(Collectors.toList());`	Create a new list that maps to the original one
`List<String> adminFirstNames = users .stream() .filter(User::isAdmin) .map(User::getFirstName) .collect(Collectors.toList());`	Combine operations This will not result in two traversals of the list elements
`int sumOfAges = users .stream() .mapToLong(User::getAge) .sum();`	Simple reduction operation
`Map<Role, List<User>> byRole = users .stream() .collect(Collectors .groupingBy(User::getRole));`	Group users by a certain attribute
`int sumOfAges = users .parallelStream() .mapToLong(User::getAge) .sum();`	All the above operations can be done in parallel

CHARACTER ESCAPE SEQUENCES

`\b`	backspace `\u0008`
`\t`	tab `\u0009`
`\n`	newline `\u000A`
`\f`	form feed `\u000C`
`\r`	carriage return `\u000D`
`\"`	double quote
`\'`	single quote
`\\`	backslash
`\uhhhh` (hhhh is a hex number between `0000` and `FFFF`)	The UTF-16 code point with value hhhh
`\ooo` (ooo is an octal number between `0` and `377`)	The character with octal value ooo

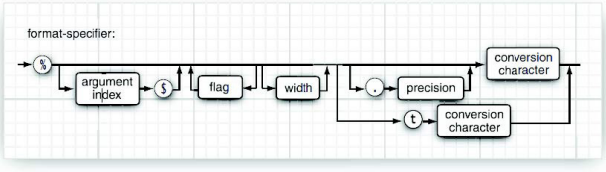
Note: Unlike in C/C++, `\xhh` is not allowed.

FORMATTED OUTPUT WITH PRINTF

TYPICAL USAGE

```
System.out.printf("%4d %8.2f", quantity, price);
String str = String.format("%4d %8.2f", quantity, price);
```

Each format specifier has the following form. See the tables for flags and conversion characters.



FLAGS

FLAG	DESCRIPTION	EXAMPLE
`+`	Prints sign for positive and negative numbers	`+3333.33`
space	Adds a space before positive numbers	`| 3333.33|`
`0`	Adds leading zeroes	`003333.33`
`-`	Left-justifies field	`|3333.33 |`
`(`	Encloses negative number in parentheses	`(3333.33)`
`,`	Adds group separators	`3,333.33`
`#` (for `f` format)	Always includes a decimal point	`3,333.`
`#` (for `x` or `o` format)	Adds 0x or 0 prefix	`0xcafe`
`$`	Specifies the index of the argument to be formatted; for example, `%1$d %1$x` prints the first argument in decimal and hexadecimal	`159 9F`
`<`	Formats the same value as the previous specification; for example, `%d %<x` prints the same number in decimal and hexadecimal	`159 9F`

CONVERSION CHARACTERS

CONVERSION CHARACTER	DESCRIPTION	EXAMPLE
`d`	Decimal integer	`159`
`x`	Hexadecimal integer	`9f`
`o`	Octal integer	`237`
`f`	Fixed-point floating-point	`15.9`
`e`	Exponential floating-point	`1.59e+01`
`g`	General floating-point (the shorter of e and f)	
`a`	Hexadecimal floating-point	`0x1.fccdp3`
`s`	String	`Hello`
`c`	Character	`H`
`b`	boolean	`true`
`h`	Hash code	`42628b2`
`tx`	Date and time	See next table
`%`	The percent symbol	`%`
`n`	The platform-dependent line separator	

FORMATTED OUTPUT WITH MESSAGEFORMAT

Typical usage:

```
String msg = MessageFormat.format("On {1, date, long}, a
{0} caused {2,number,currency} of damage.", "hurricane",
new GregorianCalendar(2009, 0, 15). getTime(), 1.0E8);
```

Yields `"On January 1, 1999, a hurricane caused $100,000,000 of damage"`

- The nth item is denoted by {n, format, subformat} with optional formats and subformats shown below
- {0} is the first item
- The following table shows the available formats

- Use single quotes for quoting, for example '{' for a literal left curly brace
- Use '' for a literal single quote

FORMAT	SUBFORMAT	EXAMPLE
number	none	1,234.567
	integer	1,235
	currency	$1,234.57
	percent	123,457%
date	none or medium	Jan 15, 2015
	short	1/15/15
	long	January 15, 2015
	full	Thursday, January 15, 2015
time	none or medium	3:45:00 PM
	short	3:45 PM
	long	3:45:00 PM PST
	full	3:45:00 PM PST
choice	List of choices, separated by |. Each choice has • a lower bound (use -\u221E for −∞) • a relational operator: < for "less than", # or \u2264 for ≤ • a message format string For example, {1,choice,0#no houses|1#one house|2#{1} houses}	no houses one house 5 houses

REGULAR EXPRESSIONS

COMMON TASKS

`String[] words = str.split("\\s+");`	Split a string along white space boundaries

`Pattern pattern = Pattern.compile("[0-9]+");`
`Matcher matcher = pattern.matcher(str);`
`String result = matcher.replaceAll("#");`
	Replace all matches. Here we replace all digit sequences with a #.

`Pattern pattern = Pattern.compile("[0-9]+");`
`Matcher matcher = pattern.matcher(str);`
`while (matcher.find()) {`
`  process(str.substring(matcher.start(), matcher.end()));`
`}`
	Find all matches.

`Pattern pattern = Pattern.compile("(1?[0-9]):([0-5][0-9])[ap]m");`
`Matcher matcher = pattern.matcher(str);`
`for (int i = 1; i <= matcher.groupCount(); i++) {`
`  process(matcher.group(i));`
`}`
	Find all groups (indicated by parentheses in the pattern). Here we find the hours and minutes in a date.

REGULAR EXPRESSION SYNTAX

CHARACTERS	
c	The character *c*
\u*nnnn*, \x*nn*, \0*n*, \0*nn*, \0*nnn*	The code unit with the given hex or octal value
\t, \n, \r, \f, \a, \e	The control characters tab, newline, return, form feed, alert, and escape
\c*c*	The control character corresponding to the character *c*
CHARACTER CLASSES	
[C_1C_2 ...]	**Union:** Any of the characters represented by $C_1C_2, \ldots$ The C_i are characters, character ranges c_1-c_2, or character classes. Example: [a-zA-Z0-9_]
[^C_1C_2 ...]	**Complement:** Characters not represented by any of $C_1C_2, \ldots$ Example: [^0-9]
[C_1&&C_2&&...]	**Intersection:** Characters represented by all of $C_1C_2, \ldots$ Example: [A-f&&[^G-`]]
PREDEFINED CHARACTER CLASSES	
.	Any character except line terminators (or any character if the DOTALL flag is set)
\d	A digit [0-9]
\D	A nondigit [^0-9]
\s	A whitespace character [\t\n\r\f\x0B]
\S	A nonwhitespace character
\w	A word character [a-zA-Z0-9_]
\W	A nonword character
\p{*name*}	A named character class—see table below
\P{*name*}	The complement of a named character class
BOUNDARY MATCHERS	
^ $	Beginning, end of input (or beginning, end of line in multiline mode)
\b	A word boundary
\B	A nonword boundary
\A	Beginning of input
\z	End of input
\Z	End of input except final line terminator
\G	End of previous match
QUANTIFIERS	
X?	Optional *X*
*X**	*X*, 0 or more times
X+	*X*, 1 or more times
X{*n*} *X*{*n*,} *X*{*n*,*m*}	*X* *n* times, at least *n* times, between *n* and *m* times
QUANTIFIER SUFFIXES	
?	Turn default (greedy) match into reluctant match
+	Turn default (greedy) match into reluctant match
SET OPERATIONS	
XY	Any string from *X*, followed by any string from *Y*
X|*Y*	Any string from *X* or *Y*

GROUPING	
(*X*)	Capture the string matching *X* as a group
\g	The match of the *g*th group
ESCAPES	
\c	The character c (must not be an alphabetic character)
\Q ... \E	Quote . . . verbatim
(? ...)	Special construct—see API notes of Pattern class

PREDEFINED CHARACTER CLASS NAMES

Lower	ASCII lower case [a-z]
Upper	ASCII upper case [A-Z]
Alpha	ASCII alphabetic [A-Za-z]
Digit	ASCII digits [0-9]
Alnum	ASCII alphabetic or digit [A-Za-z0-9]
XDigit	Hex digits [0-9A-Fa-f]
Print or Graph	Printable ASCII character [\x21-\x7E]
Punct	ASCII nonalpha or digit [\p{Print}&&\P{Alnum}]
ASCII	All ASCII [\x00-\x7F]
Cntrl	ASCII Control character [\x00-\x1F]
Blank	Space or tab [\t]
Space	Whitespace [\t\n\r\f\0x0B]
javaLowerCase	Lower case, as determined by Character.isLowerCase()
javaUpperCase	Upper case, as determined by Character.isUpperCase()
javaWhitespace	White space, as determined by Character.isWhitespace()
javaMirrored	Mirrored, as determined by Character.isMirrored()
In*Block*	*Block* is the name of a Unicode character block, with spaces removed, such as BasicLatin or Mongolian
Category or In*Category*	*Category* is the name of a Unicode character category such as L (letter) or Sc (currency symbol)

FLAGS FOR MATCHING

The pattern matching can be adjusted with flags, for example:

```
Pattern pattern = Pattern.compile(patternString,
Pattern.CASE_INSENSITIVE + Pattern.UNICODE_CASE)
```

FLAG	DESCRIPTION
CASE_INSENSITIVE	Match characters independently of the letter case. By default, this flag takes only US ASCII characters into account
UNICODE_CASE	When used in combination with CASE_INSENSITIVE, use Unicode letter case for matching
MULTILINE	^ and $ match the beginning and end of a line, not the entire input
UNIX_LINES	Only '\n' is recognized as a line terminator when matching ^ and $ in multiline mode
DOTALL	When using this flag, the . symbol matches all characters, including line terminators
CANON_EQ	Takes canonical equivalence of Unicode characters into account. For example, u followed by ¨ (diaeresis) matches ü
LITERAL	The input string that specifies the pattern is treated as a sequence of literal characters, without special meanings for . [ ] etc

LOGGING

COMMON TASKS

Logger logger = Logger.getLogger("com.mycompany.myprog.mycategory");	Get a logger for a category
logger.info("Connection successful.");	Logs a message of level FINE. Available levels are SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST, with corresponding methods severe, warning, and so on
logger.log(Level.SEVERE, "Unexpected exception", Throwable);	Logs the stack trace of a Throwable
logger.setLevel(Level.FINE);	Sets the logging level to FINE. By default, the logging level is INFO, and less severe logging messages are not logged
Handler handler = new FileHandler("%h/myapp.log", SIZE_LIMIT, LOG_ROTATION_COUNT); handler.setFormatter(new SimpleFormatter()); logger.addHandler(handler);	Adds a file handler for saving the log records in a file. See the table below for the naming pattern. This handler uses a simple formatter instead of the XML formatter that is the default for file handlers

LOGGING CONFIGURATION FILES

The logging configuration can be configured through a logging configuration file, by default `jre/lib/logging.properties`. Another file can be specified with the system property `java.util.logging.config.file` when starting the virtual machine. (Note that the `LogManager` runs before `main`.)

CONFIGURATION PROPERTY	DESCRIPTION	DEFAULT
loggerName.level	The logging level of the logger by the given name	None; the logger inherits the handler from its parent
handlers	A whitespace or comma-separated list of class names for the root logger. An instance is created for each class name, using the default constructor	java.util.logging.ConsoleHandler
loggerName.handlers	A whitespace or comma-separated list of class names for the given logger	None
loggerName.useParentHandlers	false if the parent logger's handlers (and ultimately the root logger's handlers) should not be used	true
config	A whitespace or comma-separated list of class names for initialization	None

CONFIGURATION PROPERTY	DESCRIPTION	DEFAULT
`java.util.logging.FileHandler.level` `java.util.logging.ConsoleHandler.level`	The default handler level	`Level.ALL` for `FileHandler`, `Level.INFO` for `ConsoleHandler`
`java.util.logging.FileHandler.filter` `java.util.logging.ConsoleHandler.filter`	The class name of the default filter	None
`java.util.logging.FileHandler.formatter` `java.util.logging.ConsoleHandler.formatter`	The class name of the default formatter	`java.util.logging.XMLFormatter` for `FileHandler`, `java.util.logging.SimpleFormatter` for `ConsoleHandler`
`java.util.logging.FileHandler.encoding` `java.util.logging.ConsoleHandler.encoding`	The default encoding	default platform encoding
`java.util.logging.FileHandler.limit`	The default limit for rotating log files, in bytes	0 (No limit), but set to 50000 in *jre*`/lib/logging.properties`
`java.util.logging.FileHandler.count`	The default number of rotated log files	1
`java.util.logging.FileHandler.pattern`	The default naming pattern for log files. The following tokens are replaced when the file is created: `/` Path seperator `%t` System temporary directory `%h` Value of `user.home` system property `%g` The generation number of rotated logs `%u` A unique number for resolving naming conflicts `%%` The % character	`%h/java%u.log`
`java.util.logging.FileHandler.append`	The default append mode for file loggers; `true` to append to an existing log file	`false`

PROPERTY FILES

- Contain name/value pairs, separated by `=`, `:`, or whitespace
- Whitespace around the name or before the start of the value is ignored
- Lines can be continued by placing an `\` as the last character; leading whitespace on the continuation line is ignored

```
button1.tooltip = This is a long \
                  tooltip text.
```

- `\t \n \f \r \\ \uxxxx` escapes are recognized (but not `\b` or octal escapes)
- Files are assumed to be encoded in ISO 8859-1; use `native2ascii` to encode non-ASCII characters into Unicode escapes
- Blank lines and lines starting with `#` or `!` are ignored

Typical usage:

```
Properties props = new Properties();
props.load(new FileInputStream("prog.properties"));
String value = props.getProperty("button1.tooltip");
// null if not present
```

Also used for resource bundles:

```
ResourceBundle bundle = ResourceBundle.getBundle("prog");
  // Searches for prog_en_US.properties,
  // prog_en.properties, etc.
String value = bundle.getString("button1.tooltip");
```

JAR FILES

- Used for storing applications, code libraries
- By default, class files and other resources are stored in ZIP file format
- `META-INF/MANIFEST.MF` contains JAR metadata
- `META-INF/services` can contain service provider configuration
- Use the `jar` utility to make JAR files

JAR UTILITY OPTIONS

OPTION	DESCRIPTION
`c`	Creates a new or empty archive and adds files to it. If any of the specified file names are directories, the jar program processes them recursively
`C`	Temporarily changes the directory. For example, `jar cvfC myprog.jar classes *.class` changes to the `classes` subdirectory to add class files
`e`	Creates a `Main-Class` entry in the manifest `jar cvfe myprog.jar com.mycom.mypkg.MainClass files`
`f`	Specifies the JAR file name as the second command-line argument. If this parameter is missing, jar will write the result to standard output (when creating a JAR file) or read it from standard input (when extracting or tabulating a JAR file)
`i`	Creates an index file (for speeding up lookups in a large archive)
`m`	Adds a manifest to the JAR file `jar cvfm myprog.jar mymanifest.mf files`
`M`	Does not create a manifest file for the entries
`t`	Displays the table of contents `jar tvf myprog.jar`
`u`	Updates an existing JAR file `jar uf myprog.jar com/mycom/mypkg/SomeClass.class`
`v`	Generates verbose output
`x`	Extracts files. If you supply one or more file names, only those files are extracted. Otherwise, all files are extracted `jar xf myprog.jar`
`0`	Stores without ZIP compression

COMMON JAVAC OPTIONS

OPTION	DESCRIPTION
`-cp or -classpath`	Sets the class path, used to search for class files. The class path is a list of directories, JAR files, or expressions of the form directory/'*' (Unix) or directory* (Windows). The latter refers to all JAR files in the given directory. Class path items are separated by : (Unix) or ; (Windows). If no class path is specified, it is set to the current directory. If a class path is specified, the current directory is not automatically included—add a . item if you want to include it
`-sourcepath`	Sets the path used to search for source files. If source and class files are present for a given file, the source is compiled if it is newer. If no source path is specified, it is set to the current directory
`-d`	Sets the path used to place the class files. Use this option to separate `.java` and `.class` files
`-source`	Sets the source level. Valid values are 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 5, 6, 7, 8
`-deprecation`	Gives detail information about the use of deprecated features
`-Xlint:unchecked`	Gives detail information about unchecked type conversion warnings

COMMON JAVA OPTIONS

OPTION	DESCRIPTION
`-cp or -classpath`	Sets the class path, used to search for class files. See the previous table for details. Note that `javac` can succeed when `java` fails if the current directory is on the source path but not the class path
`-ea or -enableassertions`	Enable assertions. By default, assertions are disabled
`-D`*`property=value`	Sets a system property that can be retrieved by `System.getProperty(String)`
`-jar`	Runs a program contained in a JAR file whose manifest has a `Main-Class` entry. When this option is used, the class path is ignored
`-verbose`	Shows the classes that are loaded. This option may be useful to debug class loading problems
`-Xms`*`size`* `-Xmx`*`size`*	Sets the initial or maximum heap size. The size is a value in bytes. Add a suffix `k` or `m` for kilobytes or megabytes, for example, `-Xmx10m`

ABOUT THE AUTHORS

Cay S. Horstmann has written many books on C++, Java and objectoriented development, is the series editor for Core Books at Prentice-Hall and a frequent speaker at computer industry conferences. For four years, Cay was VP and CTO of an Internet startup that went from 3 people in a tiny office to a public company. He is now a computer science professor at San Jose State University. He was elected Java Champion in 2005. Cay blogs at weblogs.java.net/blog/cayhorstmann.

Ivan St. Ivanov is a development architect at SAP Labs Bulgaria, working in the HANA Cloud Platform performance team. He is a leader in the Bulgarian JUG, driving the adoption of OpenJDK in Bulgaria. In his free time he likes contributing to open-source software, mostly to JBoss Forge. Ivan is obtaining his doctorate in the area of cloud multi-tenancy at the University of National and World Economy in Sofia. He is teaching Java and Java EE at Sofia University.

CREDITS:

Editor: **G. Ryan Spain** | Designer: **Yassee Mohebbi** | Production: **Chris Smith** | Sponsor Relations: **Chris Brumfield** | Marketing: **Chelsea Bosworth**

DZone

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, research guides, feature articles, source code and more.

"DZone is a developer's dream," says PC Magazine.

DZONE, INC.
150 PRESTON EXECUTIVE DR.
CARY, NC 27513

888.678.0399
919.678.0300

REFCARDZ FEEDBACK WELCOME
refcardz@dzone.com

SPONSORSHIP OPPORTUNITIES
sales@dzone.com

ISBN-13: 978-1-936502-77-6
ISBN-10: 1-936502-77-1
50795

VERSION 1.0 **$7.95**