Java CLI JDK command line

START HERE

`java Hello.java`
Compile and run one file

`javac -d out src/*.java`
Compile to a folder

`java -cp out Main`
Run a class

`jar --create --file app.jar --main-class Main -C out .`
Build a runnable JAR

`java -jar app.jar`
Run a JAR

`java -Xmx2g -jar app.jar`
Cap the heap

`jshell`
Try code interactively

`jcmd`
List running JVMs

Main.java source → `javac` → **Main.class** bytecode → `jar` → **app.jar** archive → `java -jar` → **JVM** running app

CONCEPTS

Source becomes bytecode with javac, bytecode runs on the JVM with java. Tools like jar, jshell and jcmd all ship in the JDK.

Class path separator is : on Linux and macOS, ; on Windows.

A JRE-only install has java but no javac, jar or jshell: install a JDK.

RUN

`java -version`
Print version and exit
Goes to stderr; --version prints to stdout.

`java <File>.java [args]`
Compile in memory and run
Runs the first class in the file; it needs a main method.

`java -cp <path> <MainClass> [args]`
Run a class

`java -jar <file.jar> [args]`
Run a JAR's main class
-cp is ignored here; use the manifest Class-Path.

`java -Dkey=value ...`
Set a system property
Read it with System.getProperty("key").

`java -ea ...`
Enable assertions

`java -m <module>/<MainClass>`
Run a module

`java --enable-preview ...`
Allow preview features

`java @<argfile>`
Read options from a file

CLASS PATH & MODULES

`-cp "lib/*:out"`
Add all JARs in lib, plus out
Quote it so the shell does not expand the star.

`--module-path <dir> / -p <dir>`
Where application modules live

`--add-modules <mod>[,<mod>]`
Add root modules

CLASS PATH & MODULES (cont.)

`--add-opens <mod>/<pkg>=ALL-UNNAMED`
Open a package to reflection
Fixes InaccessibleObjectException on JDK 16+.

`--add-exports <mod>/<pkg>=ALL-UNNAMED`
Export a package

`java --list-modules`
List observable modules

MEMORY & JVM FLAGS

`-Xmx<size>`
Maximum heap
Suffixes k, m, g: -Xmx512m, -Xmx2g.

`-Xms<size>`
Initial heap

`-Xss<size>`
Thread stack size
Raise it on StackOverflowError in deep recursion.

`-XX:MaxRAMPercentage=<n>`
Heap as share of RAM
Handy in containers instead of a fixed -Xmx.

`-XX:+HeapDumpOnOutOfMemoryError`
Write a heap dump on OOM

`-XX:HeapDumpPath=<file>`
Where the dump goes

`-verbose:gc`
Log garbage collections

COMPILE (JAVAC)

`javac <File>.java`
Compile one file

`javac -d <dir> <files>`
Put class files in dir
The directory is created if it is missing.

`javac -cp <path> <files>`
Compile against libraries

`javac --release <n> <files>`
Target a Java version
Also checks against that release's API.

`javac -Xlint:all <files>`
Enable all warnings

`javac -encoding UTF-8 <files>`
Source file encoding

`javac -sourcepath <dir> <File>.java`
Find other sources

COMPILE (JAVAC) (cont.)

`javac -parameters <files>`
Keep parameter names

`javac @<argfile>`
Options and files from a file

PACKAGE (JAR)

`jar --create --file <f.jar> -C <dir> .`
Pack a directory

`jar --create --file <f.jar> --main-class <Class> -C <dir> .`
Runnable JAR
Writes Main-Class into META-INF/MANIFEST.MF.

`jar -tf <f.jar>`
List contents

`jar -xf <f.jar> [<entry>]`
Extract all or one entry

`jar --update --file <f.jar> -C <dir> <file>`
Add or replace a file

`jar --create --file <f.jar> --manifest <file> -C <dir> .`
Use your own manifest

JSHELL

`jshell`
Start the REPL
Leave with /exit.

`jshell --class-path <path>`
Start with libraries

`/vars`
Show variables

`/methods`
Show methods

`/list`
Show snippets with IDs

`/save <file>`
Save snippets to a file

`/open <file>`
Load a script

`/help`
List commands

Shift+Tab then v turns an expression into a variable; Shift+Tab then i suggests imports.

DIAGNOSE

`jcmd`
List running JVMs

DIAGNOSE (cont.)

`jps -l`
PIDs with main class or JAR

`jcmd <pid> Thread.print`
Thread dump
Add -l for lock information.

`jstack <pid>`
Thread dump (older tool)

`jcmd <pid> GC.heap_info`
Heap usage summary

`jcmd <pid> GC.class_histogram`
Objects per class

`jcmd <pid> GC.heap_dump <file>`
Write a heap dump
Slow and large; use -gz=1 to compress.

`jcmd <pid> VM.flags`
Flags of the running JVM

`jcmd <pid> VM.system_properties`
System properties

`jcmd <pid> JFR.start duration=60s filename=<f.jfr>`
Record a flight recording

ENVIRONMENT

`JAVA_HOME`
JDK install directory

`JAVA_TOOL_OPTIONS`
Options for every JVM
Picked up by all JVMs, including build tools; the JVM prints a notice.

RECIPES

Build and run a small app

```
javac -d out $(find src -name '*.java')
jar --create --file app.jar --main-class com.example.Main -C out .
java -jar app.jar
```

Run with libraries

```
javac -cp "lib/*" -d out src/Main.java
java -cp "out:lib/*" Main
```

Diagnose a stuck app

```
jps -l
jcmd <pid> Thread.print -l > threads.txt
jcmd <pid> GC.heap_info
```

Capture memory failures

```
java -Xmx512m \
  -XX:+HeapDumpOnOutOfMemoryError \
  -XX:HeapDumpPath=/tmp/heap.hprof \
  -jar app.jar
```